21世纪高等学校规划教材 | 软件工程





软件工程

鄂大伟 主编

清华大学出版社

21世纪高等学校规划教材·软件工程

软 件 工 程

鄂大伟　主编
尤志宁　叶文来　蔡莉白　易　燕　等编著

清华大学出版社
北　京

内 容 简 介

本书在软件工程的知识域组织方面参考了 IEEE 和 ACM 提出的“软件工程的知识体系(SWEBOK)”的基本框架,比较全面、系统地反应了软件工程的全貌,从理论与实践的视角介绍了软件工程的基本原理、概念和技术方法。全书共 18 章,在内容结构上可分为软件工程与项目管理、结构化开发方法、面向对象的开发方法及软件工程高级专题 4 个部分。在每章后面都附有思考与练习题,供读者复习巩固之用。

本书汲取了国内外软件工程的精华,并融入了作者多年在教学与科研过程中对软件工程的理解与经验总结。在内容上既兼顾了传统、实用的软件开发方法,又引入了软件工程领域比较新颖的技术和方法,并结合一个贯穿全书的具体案例加以介绍。本书的另一个特点是介绍了许多软件工程度量与估算的技术与方法,突出了软件工程学科工程化、可度量的特点。

本书可作为计算机相关专业本科生或研究生的教材,同时也可作为软件工程领域专业人员的参考书。

图书在版编目(CIP)数据

软件工程/鄂大伟主编;尤志宁,叶文来,蔡莉白,易燕等编著. —北京:清华大学出版社,2010.8
(21 世纪高等学校规划教材·软件工程)
ISBN 978-7-302-22680-2

Ⅰ. ①软… Ⅱ. ①鄂… ②尤… ③叶… ④蔡… ⑤易… Ⅲ. ①软件工程—高等学校—教材 Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2010)第 084983 号

责任编辑: 索 梅 薛 阳
责任校对: 梁 毅
责任印制:

出版发行: 清华大学出版社 **地 址:** 北京清华大学学研大厦 A 座
http://www.tup.com.cn **邮 编:** 100084
社 总 机: 010-62770175 **邮 购:** 010-62786544
投稿与读者服务: 010-62795954,jsjjc@tup.tsinghua.edu.cn
质 量 反 馈: 010-62772015,zhiliang@tup.tsinghua.edu.cn
印 刷 者:
装 订 者:
经 销: 全国新华书店
开 本: 185×260 **印 张:** 29.25 **字 数:** 705 千字
版 次: 2010 年 8 月第 1 版 **印 次:** 2010 年 8 月第1 次印刷
印 数: 1～ 000
定 价: .00 元

产品编号:

编审委员会成员

（按地区排序）

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施“高等学校本科教学质量与教学改革工程(简称‘质量工程’)”，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部“质量工程”的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程(群)加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合“质量工程”的实施，满足各高校教学质量和教学改革的需要。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作，提高教学质量的若干意见》精神，紧密配合教育部已经启动的“高等学校教学质量与教学改革工程精品课程建设工作”，在有关专家、教授的倡议和有关部门的大力支持下，我们组织并成立了“清华大学出版社教材编审委员会”(以下简称“编委会”)，旨在配合教育部制定精品课程教材的出版规划，讨论并实施精品课程教材的编写与出版工作。“编委会”成员皆来自全国各类高等学校教学与科研第一线的骨干教师，其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求，“编委会”一致认为，精品课程的建设工作从开始就要坚持高标准、严要求，处于一个比较高的起点上；精品课程教材应该能够反映各高校教学改革与课程建设的需要，要有特色风格、有创新性(新体系、新内容、新手段、新思路，教材的内容体系有较高的科学创新、技术创新和理念创新的含量)、先进性(对原有的学科体系有实质性的改革和发展，顺应并符合 21 世纪教学发展的规律，代表并引领课程发展的趋势和方向)、示范性(教材所体现的课程体系具有较广泛的辐射性和示范性)和一定的前瞻性。教材由个人申报或各校推荐(通过所在高校的“编委会”成员推荐)，经“编委会”认真评审，最后由清华大学出版

社审定出版。

目前,针对计算机类和电子信息类相关专业成立了两个“编委会”,即“清华大学出版社计算机教材编审委员会”和“清华大学出版社电子信息教材编审委员会”。推出的特色精品教材包括:

(1) 21世纪高等学校规划教材·计算机应用——高等学校各类专业,特别是非计算机专业的计算机应用类教材。

(2) 21世纪高等学校规划教材·计算机科学与技术——高等学校计算机相关专业的教材。

(3) 21世纪高等学校规划教材·电子信息——高等学校电子信息相关专业的教材。

(4) 21世纪高等学校规划教材·软件工程——高等学校软件工程相关专业的教材。

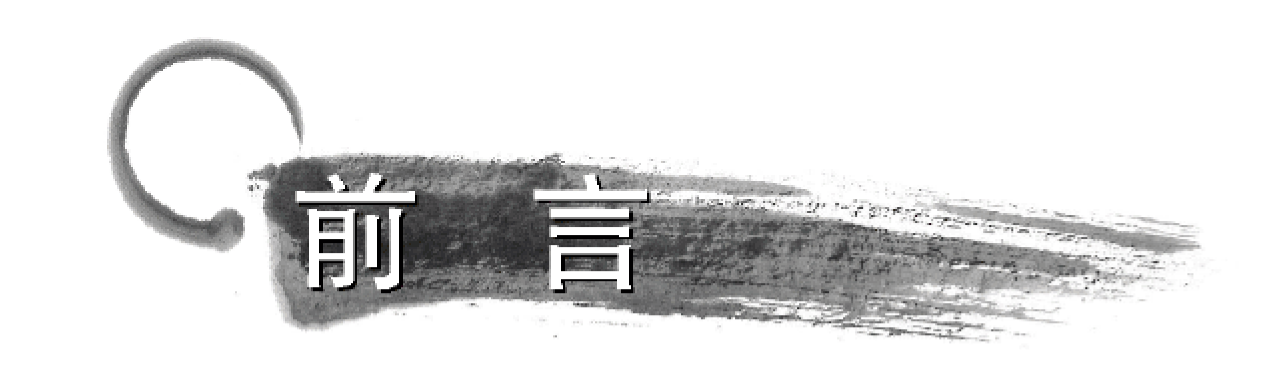
(5) 21世纪高等学校规划教材·信息管理与信息系统。

(6) 21世纪高等学校规划教材·财经管理与计算机应用。

(7) 21世纪高等学校规划教材·电子商务。

清华大学出版社经过二十多年的努力,在教材尤其是计算机和电子信息类专业教材出版方面树立了权威品牌,为我国的高等教育事业做出了重要贡献。清华版教材形成了技术准确、内容严谨的独特风格,这种风格将延续并反映在特色精品教材的建设中。

清华大学出版社教材编审委员会
联系人:魏江江
E-mail:weijj@tup.tsinghua.edu.cn



软件系统现在已经变得无处不在，被广泛应用于各个领域，深入到生活的各个方面。事实上，没有软件就没有计算机，生活中也就没有手机、因特网和各种智能家电，航天工程与太空探索更成为天方夜谭。而所有这些软件系统的描述、设计、开发和管理就构成了软件工程的基本内容。

与其他传统工程学科相比，软件工程还是一个较为年轻的学科。然而，自 20 世纪 60 年代提出软件工程的概念以来，在这个领域已经取得长足的进步，提出了许多软件工程的方法、过程与工具，这些成果已经极大地改善了软件的开发方式，软件开发活动与项目管理也有了更规范的指导。目前，软件产业已发展为国家基础性、先导性、战略性产业，成为信息产业、先进制造业和现代服务业的核心。据《2009 中国软件与信息服务外包产业发展报告》显示，2008 年我国软件与信息服务外包产业规模达到 1570 亿元。此外，2008 年软件与信息服务业规模已达到 7000 亿元，软件产业规模居世界第 4 位。软件的地位越来越重要。

软件项目开发实践表明，即使是最简单的软件，系统也有其固有的复杂性，因此，必须在软件开发中使用工程原则。软件工程是这样一个工程学科，即软件工程师用计算机科学中的方法和理论，在考虑成本效益的情况下，将其用于解决软件开发中的复杂问题。不仅大型软件项目需要运用软件工程的概念、原则和方法，就是一般的小型软件项目也必须掌握和运用软件工程知识，才能适应工作的需要。软件开发的工程实践从另一个方面也表明，不掌握软件工程知识，不按照软件工程的方法与过程管理软件项目，提供给用户的产品只能是低质量的、难以满足用户需求的。事实上，绝大多数的现代软件能为用户提供好的服务，我们不应该因某些软件项目的失败而无视过去几十年软件工程领域的巨大成功。

“软件工程”是高等院校计算机及相关专业教学计划中的一门核心专业课程。本书在知识域组织方面参考了 IEEE 和 ACM 提出的“软件工程的知识体系”(SWEBOK)的基本框架，比较全面、系统地反映了软件工程的全貌，从理论与实践的视角介绍了软件工程的基本原理、概念和技术方法。本书内容既兼顾了传统、实用的软件开发方法，又介绍了软件工程领域比较新颖的技术和方法，并结合具体案例加以介绍，其中融入许多我们在多年教学中对软件工程的理解与经验总结，努力使之成为软件工程的原理、方法和应用紧密结合的教材。

本书共 18 章，从内容上可分为 4 个部分。

第 1 篇　软件工程与项目管理(第 1～第 6 章)

作为全书的首篇，前两章主要介绍了软件工程的基本概念和软件过程模型，第 3 章～第 6 章，参照美国项目管理学会(PMI)提出的项目管理框架，结合软件工程的实践，分别叙述了软件项目管理的主要活动，包括项目沟通(需求获取)、软件项目计划、项目范围、项目估算、进度管理、软件质量保证、软件过程能力评估、软件配置管理、风险管理等内容。本篇还介绍了软件工程经济学中一些最基本的内容，可以帮助项目管理者对项目状态进行成本效益分析与决策。

第 2 篇　结构化开发方法(第 7～第 10 章)

结构化开发方法经过多年的发展,已经形成了一套比较成熟的理论。本篇介绍了结构化分析与设计方法,以数据流图为基础,通过 ATM(自动取款机)建模实例,逐层划分出流图,以得到系统的软件结构,并将其映射为软件功能。本篇还介绍了结构化的度量方法。

第 3 篇　面向对象的开发方法(第 11～第 15 章)

面向对象方法学把对象作为融合了数据及在数据上的操作行为的统一软件构架,用对象分解取代了传统方法的功能分解。近年来在许多应用领域中面向对象方法学已经迅速取代了传统的方法学。本篇较为全面、系统、清晰地介绍了面向对象软件工程的基本概念、原理、方法和工具,通过 ATM 实例,结合 UML 进行可视化建模,说明了面向对象软件开发的整个过程,同时介绍了面向对象的度量方法。

第 4 篇　软件工程高级专题(第 16～第 18 章)

本篇内容包括敏捷过程开发、Web 工程、形式化开发方法,并通过实例加以介绍。之所以称之为"高级专题",是因为这些内容有助于扩展对软件工程的深入理解,有些技术可能代表着软件工程的未来发展方向,并对软件技术产生影响。

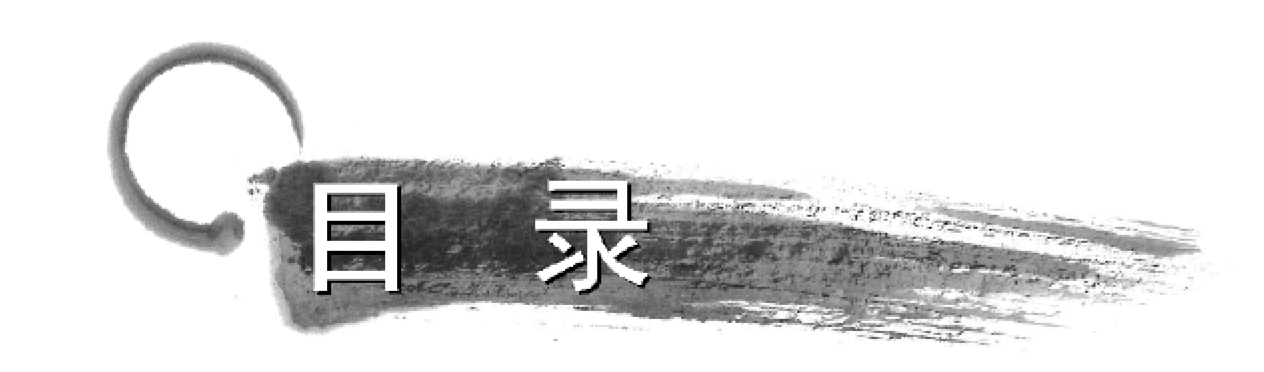
参与本书编写的教师是福建省省级精品课程"软件工程"的课程负责人及课程团队成员,他们长期从事"软件工程"及相关课程的教学与科研,具有较深厚的软件基础理论和丰富的软件开发实践经验。本书由鄂大伟教授主编并编写第 1 章～第 6 章的内容,尤志宁编写第 7 章～第 9 章,叶文来编写第 11 章～第 14 章,蔡莉白编写第 10 章、第 15 章、第 16 章和第 18 章的内容,易燕编写第 17 章的内容。

软件工程是一门处于前沿地位的重要学科,在日新月异的计算机科学与技术的风帆鼓动下迅速地前进。希望读者能通过本书的学习,将理论知识应用于软件开发的实践,并不断地创新,为今后研究这门学科奠定良好的基础。

由于作者水平有限,书中疏漏、欠妥之处在所难免,恳请读者批评指正。

作　者

2010 年 2 月于厦门集美学村



第1篇　软件工程与项目管理

第 4 篇 软件工程高级专题

第1篇 软件工程与项目管理



本篇内容：

- 软件工程概述
- 软件过程
- 软件工程领域下的项目管理
- 软件项目估算
- 软件质量管理
- 软件风险管理

第1章 软件工程概述

软件工程是一种方法论，而不是一种具体的摸得着、看得见的产品。

——《软件工程——原理、方法与应用》

软件工程（Software Engineering）是将系统化的、规范的、可度量的方法应用于软件的开发、运行和维护的工程。它应用计算机科学、数学及管理科学等原理，借鉴系统工程的原则、方法，构建软件以达到提高质量、降低成本的目的。多年来，人们围绕着实现生产高质量的软件这个目标，从技术到管理做了大量的努力，逐渐形成了“软件工程学”这一计算机新学科。

软件工程学所研究的主要内容包括软件开发技术和软件工程管理两个方面。其中，软件开发技术包含了软件开发学、软件工具和软件工程环境；软件工程管理包含了软件管理学、软件经济学和软件度量学。由于软件的特殊性，软件工程不同于传统工程，它更强调抽象、建模、信息组织和表达以及管理。

1.1 软件的定义与特点

1.1.1 什么是软件

本书开篇明义，首先要回答的第一个问题就是：什么是软件，它与其他客观事物有什么不同的特点？这个问题似乎不难回答，因为软件遍布世界。环顾我们生活与办公的场所，到处都有软件的影子。人们每天使用的手机、汽车、数字电视、微波炉、智能玩具都有嵌入式软件的身影，更不用说计算机中安装的各式各样的软件了。事实上，现在软件或者显式地或者在幕后为人们生活的各方面服务，包括在现代通信、商务处理、工业控制、宇宙探索等方面发挥出巨大的威力，对整个社会的经济和文化产生了深远的影响。然而，真正对软件下一个合适的定义却是很困难的事情。

在软件的发展过程中，软件从个性化的程序演变为工程化的产品，人们对软件的看法发生了根本性的变化。“软件＝程序”显然不能涵盖软件的完整内容，除了程序之外，软件还包括与之相关的文档和配置数据，以保证这些程序的正确运行。

关于软件，教科书上一般是这样定义的：软件是：①能够完成预定功能和性能的可执行的指令(计算机程序)；②使程序能够适当地操作信息的数据结构；③描述程序的操作和使用的文档。

从广义上讲，软件是依据某一特定的观念、原则所形成的某类操作行为和文件。计算机

软件，则是指计算机各操作程序、操作程序所使用的数据以及有关的文档资料的集合[①]。

然而，软件的真正含义却不是一个形式的定义所能体现的。从软件的内容来看，软件更像是一种嵌入式的数字化知识，其形成是一个通过交互对话和抽象理解而不断演化的过程。软件的应用领域十分广泛，呈现形式也是多种多样的，在某种程度上很难对软件的类型给出一个通用的界定。

1.1.2 软件的特点与本质

从软件的一般属性和本质来看，具有如下几个特点。

1. 软件是一种逻辑实体，不是具体的物理实体

在传统工程学的范围内，工程师所致力的对象是自然界的物质或运用已经知道其规律的物理现象。没有什么实体可以和软件这种人造"产品"的情况相提并论。计算机软件是一种逻辑产品，它与物质产品有很大的区别，软件具有抽象性。电磁物理载体对软件的承载是一种状态的形式性承载，而非物质材料的内容性承载[②]。人们可以把它记录在纸、内存、磁盘和光盘等各种存储介质上，但却无法看到软件本身的形态，必须通过观察、分析、思考和判断，才能了解它的功能、性能等特性。这说明，软件产品是脑力劳动的结晶，总是以程序和文档的形式出现，通过计算机的运行才能体现它的功能和作用。软件显现与软件是两种不同的存在（软件经由运行显现出的结果被称为软件显现）。

2. 软件的不可见性决定了它的抽象性

抽象是从众多的事物中抽取出共同的、本质性的特征，而舍弃其非本质的特征。抽象是一种在有效归纳层次上对问题进行描述的过程，帮助人们专注于问题的关键方面，而不至于陷于细节。

一般情况下，人们可以通过几何抽象模型准确地描述有形的物体，例如，建筑师可以用平面图描述建筑物的结构，硬件工程师可以用电路图描述计算机的系统结构，甚至化学家也可以用分子模型描述客观事物的微观结构。但是，软件是客观世界空间和计算机空间之间的一种逻辑实体，不具有物理的形体特征。人们一直试图用不同的图形技术来描述软件结构，即便是现在流行的面向对象技术，仍然无法对它进行准确的、完整的描述。

由于软件的不可见性，定义"需要做什么"成为软件开发的根本问题。过去，几乎所有的方法、工具和过程（又称为软件工程的三要素）都致力于解决"怎么做"这个次要问题，并且取得了很大的进展，但是在"做什么"的问题上，几十年来情况似乎并没有太大的改善。

3. 软件的生产是一种认知过程

程序就是"直接可执行的想法"。这种提法是一个全新概念[③]。软件设计（编写程序）就

---

① 软件标准化与质量工作组. 我国软件标准化的发展战略与对策[J]. 中国标准化，1997，9

② 卫红春，等. 软件概念的哲学意蕴[J]. 科学技术与辩证法，2006，23(6)：22～26

③ 普林茨. 软件工程. 金维克，译. 北京：科学出版社，2005

是用自动的方法将准备表达和操作的实体从真实世界中提炼和抽象出来，其行为必须与程序所取代的真实世界中的实体相同。按照这种观点，程序设计就是构建一种符号，这个符号的位置可以在人与人之间、在人与系统之间，也可以在系统和系统之间。这样才有助于在社会的所有实体(人、组织和信息系统)之间交换信息，但同时也受到各种限制，特别是受到来自外部环境方面的限制。所以它必须同步演变，以求得软件的生命周期的延续。在这个过程中，软件离不开对客观世界问题的认识，软件参与着认识，帮助和深化着认识，软件也在反映着认识。

传统的软件实体观点把人们对软件的理解导向简单化表面化。实际上，软件具有其深刻的哲学蕴涵。从认识论角度看，软件是认识的中介，是对人的部分意识程式、动态复现式的数字化外化。软件的意识外化有客观性、符号承载性、数字化、可计算性、动态复现性和类主体性等特征。

软件开发本身就是一个认知活动。在软件开发过程中，开发者是认识的主体，软件服务的业务领域和软件需求是认识的客体。认识主体需要对客体经过从感性到理性多次反复地认识，并运用创造性思维，把用户需求转化为软件模型，最后产生出能够解决特定问题的软件系统。另外，计算机是帮助人们认识的中介和工具，也是对人认识的承载、反映和复现。

软件的内容实际上全部是人的认识内容，是对人的认知的一种描述、承载和复现。从这个角度来认识软件，人们产生了基于反映的软件认识观，这种软件认识观是对软件本质的、全面的反映。因此，计算机是通过软件来承载和反映人的认知。

4. 软件的构造性与演化性

软件是对客观世界中问题空间与解空间的具体描述，是客观事物的一种反映，是知识的提炼和“固化”。客观世界是不断变化的，因此，构造性和演化性是软件的本质特征。如何使软件模型具有更强的表达能力，更符合人类的思维模式，即如何提升计算环境的抽象层次，在一定意义上来讲，这紧紧围绕了软件的本质特征——构造性和演化性。

在各种编程语言中，广泛使用了变量、标识符、表达式等概念作为语言的基本构造，并使用 3 种基本控制结构(顺序、分支、循环)来表达软件模型的计算逻辑，使得软件开发人员可以在一个更高的抽象层次上进行程序设计。随后出现了一系列开发范型和结构化程序设计技术，实现了模块化的数据抽象和过程抽象，提高了人们表达客观世界的抽象层次，并使开发的软件具有一定的构造性。

由于软件是人类思维和智能的一种延伸，因此当软件被真正应用之后，人们往往希望超越原有的应用域范围进行软件功能的提升或扩展。另外，由于软件必须依附于硬件平台，因此需要随着硬件设备的更新和接口的不同而变化，随着时间的推移，当改善软件功能的成本变得难以接受时，软件就被“废弃”。这种需要随着应用需求、硬件、用户和社会等各种因素的变化而不断地被修改和扩展的特性表现了软件的演化性。

5. 软件的非实体性

软件是一种特殊的智力产品，是纯粹思维活动的产物，与其他物质产品相比，其最重要的特点是使用过程中不存在功能和性能损耗。物质事物的可毁灭性和软件的性质也有本质的区别。就一般实体物质而言，废弃后本身并没有消失，只是存在形式发生了改变(如锈蚀

的钢铁)。但软件一旦被“废弃”或删除,不是物质形式的变化,而是消失得无影无踪,不残存任何痕迹。

图 1-1 给出了硬件的失效率曲线,它是一个 U 形曲线,随时间的增加,失效率急剧上升。图 1-2 所描述的软件失效率曲线,它没有“U 形”曲线的右半段,表明软件随着使用时间的增加,失效率降低;因为软件不存在磨损和老化问题,但存在退化问题。

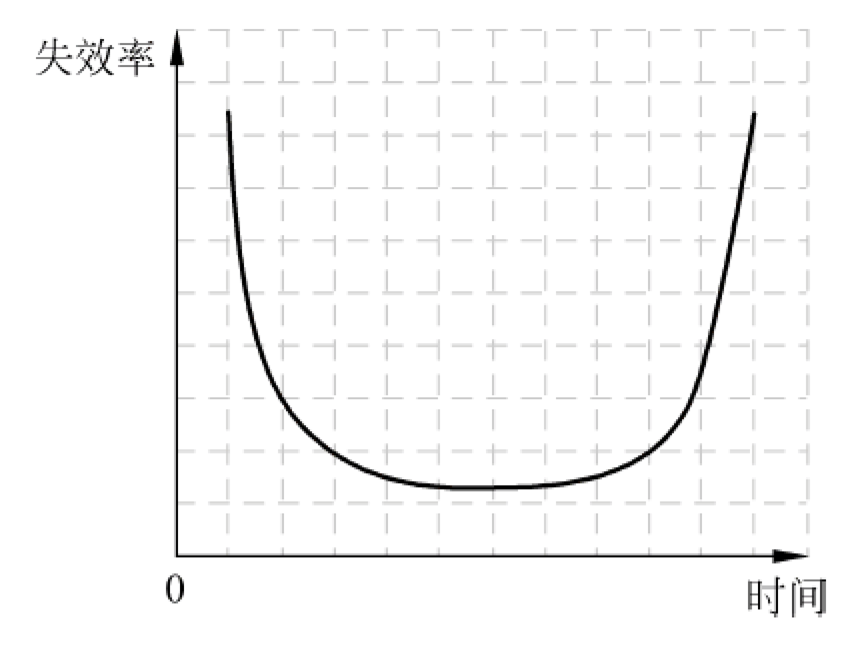


图 1-1 硬件失效率曲线

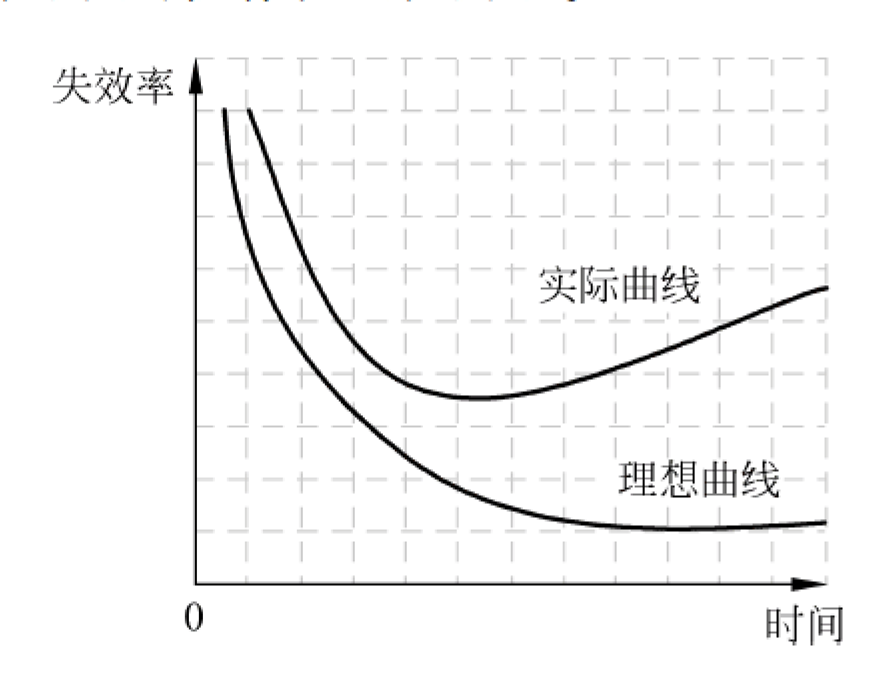


图 1-2 软件失效率曲线

关于磨损的另一个侧面也表明了硬件和软件之间的不同。当一个硬件构件磨损时,可以用另外一个备用零件替换它,但对于软件就没有备用零件可以替换了。每一个软件故障都表明了设计或是将设计转换成机器可执行代码的过程中存在错误。因此,软件维护要比硬件维护复杂得多。

6. 软件的本质是数字存在

突破软件实体性定义的局限,转向认识论,我们会对软件形成全新的认识。作为一种特定的客观存在,软件具有其本体论意义(本体论是关于一般存在和存在本身的哲学学说)。软件具有物质的客观实在性,但又具有一般物质事物所不曾具有的特殊存在性,软件的广延性和可删除性与一般物质事物具有本质的区别。软件不能独立存在,需要载体承载。软件的载体有大脑意识载体、语言符号载体和电磁物理载体 3 种形式,软件在其电磁物理载体上表现为驻存型、执行型和传输型 3 种存在形态。软件只有经过数字化才能取得自己的存在地位,软件的本质是数字存在[①]。

1.1.3 “没有银弹”——复杂性是“软件危机”的本质原因

20 世纪 60 年代末期,计算机程序在复杂度、规模和应用领域等方面的增长引人注目,这导致大量资金花费在软件开发上,许多人的工作和生活依赖于软件开发的成果。软件产品帮助人们获得更高的工作和生产效率,同时也给人们提供一个更加安全、灵活和宽松的工作与生活环境。尽管有很多成功之处,许多软件产品在成本、工期、质量等方面还是存在着严重问题。主要原因是:①软件产品是复杂的人造系统,具有复杂性、不可见性和易变性,难以处理;②个人或小组开发小型软件时具有非常有效的编程技术和过程,在开发大型、复杂系统时难以发挥同样的作用。

---

① 卫红春,等.软件概念的哲学意蕴[J].科学技术与辩证法,2006,23(6):22～26

“软件危机”的概念是在1968年北大西洋公约组织(NATO)的计算机科学家在联邦德国召开的国际学术会议上首次提出的(图1-3)。软件开发长期以来存在“开发周期长、成本高、质量差、适应性差、难维护”这五大难题，在早期人们称它为“软件危机”，它是计算机科学发展进程的必然产物，只不过到后来这种现象日渐严重，已经影响到计算机事业的发展，因而才引起各界的关注。

软件危机表现最突出的实例是美国IBM公司在1963—1966年开发的IBM 360机的操作系统。该项目花费了5000人一年的工作量，最多时有1000人投入开发工作，写了近100万行源代码。尽管投入了这么多的人力物力，得到的结果却非常糟。据统计，这个操作系统每次发行的新版本都是从前一版本中找出1000个程序错误而修正的结果。当时的糟糕情景正像这个项目负责人布鲁克斯(F. P. Brooks)后来在他所写的著作《人月神话》中描述的那样(图1-4)[①]：“……正像一只逃亡的野兽落到泥潭中做垂死的挣扎，越是挣扎，陷得越深。最后无法逃脱灭顶的灾难……程序设计工作正像这样一个泥潭……一批批程序员被迫在泥潭中拼命挣扎……谁也没有料到问题竟会陷入这样的困境……”。



图1-3 在NATO会议首次提出了软件危机的概念(1968年)



图1-4 《人月神话》插图

为解决软件开发中存在的这些问题，NATO会议上还首次提出“软件工程”(Software Engineering)的概念，提出把软件开发从“艺术”和“个体行为”向“工程”和“群体协同工作”转化。其基本思想是应用计算机科学理论和技术以及工程管理原则和方法，按照预算和进度，实现满足用户要求的软件产品的定义、开发、发布和维护的工程。1972年IEEE(国际电气与电子工程师协会)的计算机协会第一次出版了“软件工程学报”。此后，“软件工程”这个术语被广泛应用于工业、政府和学术界，众多的出版物、团体和组织、专业会议在它们的名称里使用“软件工程”这个术语，很多大学的计算机科学系先后设立了软件工程课程。

Pressman对软件危机有不同的看法，在其著作《软件工程——实践者的研究方法》中指出：我们已经到了计算机软件的危机阶段，实际上我们真正的问题应该是一种“慢性的苦恼”。“慢性的”这个词的定义能够更加贴切地反映问题：持续很长时间或经常重犯；不确定地延续。这很准确地描述了过去几十年人们所经历的问题是一种慢性的苦恼，而不是危机，并没有一种灵丹妙药可以完全治愈这种病痛，但在人们正努力去发现解决方案的同时会

① F. P. Brooks. The Mythical Man-Month

得到很多缓解痛苦的方法。

无法解除危机是因为软件学科固有的本性,在规模上,软件实体可能比任何由人类创造的其他实体要复杂,因为没有任何两个软件部分是相同的。当它执行时,软件经过一种不连续的离散状态,一个"比特"位的变化就会导致整个软件状态的变化。而这种软件状态的总数量是惊人的。这使得构思、描述和测试都非常困难。软件系统的状态比计算机系统状态多若干个数量级①。

软件系统本身是一个复杂系统,事实上,软件的复杂性来自它所反映的实际问题的复杂性。软件费用不断增加,软件规模不断庞大,软件环境日益复杂,程序逻辑结构的复杂性等,带来了许多复杂性问题。人的因素往往成为软件开发的难点,直接影响到软件项目的成败,而且软件的维护也比硬件维护复杂得多。

Brooks 分别于 1975 年和 1987 年出版了两本著名的软件工程知识著作《人月神话》和《没有银弹》(《No Silver Bullet》)。Brooks 的著名论断是"软件工作是人类所从事的最复杂的工作"。在他看来,对于软件的突破性技术(即所谓的银弹)而言,不管是过去、现在还是将来,类似于"银弹"的各种"屠龙之技"都不可能解决软件复杂性的问题(图 1-5)。这让人们意识到了软件工程的本质,放弃寻找"放之四海而皆准"的幻想。

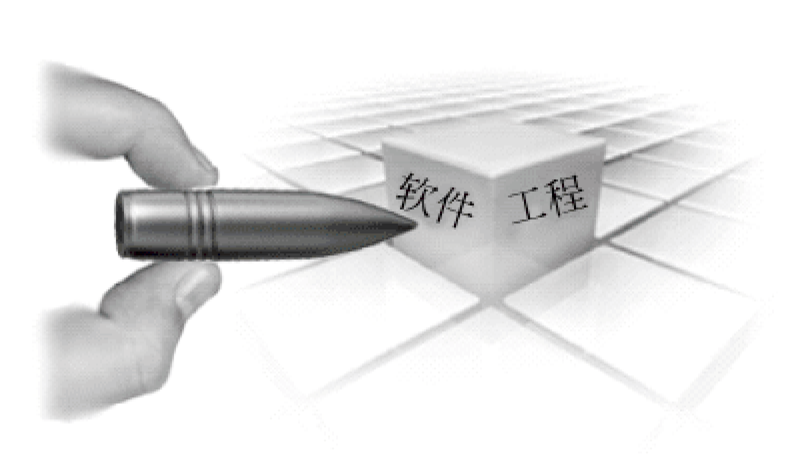


图 1-5 "没有银弹"的软件工程

即使在著名的微软公司,也经历了这种"危机",并称之为微软公司历史上第二次大灾难。原定于 1986 年 7 月与公众见面的 Macintosh 版 Word 3.0,千呼万唤方于 1987 年 2 月问世,而且先天不足,里面竟然有大约 700 多处错误——有的甚至破坏数据,摧毁程序。一下子微软公司便名声扫地了。微软公司不得不在初始发布后的两个月内免费为消费者提供升级版本,其费用超过了 100 万美元。

微软公司的测试主管罗格·舍曼,回忆了微软公司历史上这一段黯然无光的日子说:

"人们得到消息说他们实际上把事情弄得一团糟……他们像驾着一辆飞驰的赛车,不时地撞到墙上,不过现在他们总算知道墙在哪儿了……人们认识到不管程序是多么的充满想象力或创造力,如果不实用,也只能忍痛割爱。……终于,有人得出结论,说所谓的出品时间统统都是不现实的,项目也凌乱不堪,我们永远不可能及时推出产品。"②

即使是最简单的软件,系统也有其固有的复杂性,这些复杂问题的存在意味着许多软件开发项目的进行并不总是成功的。因此,必须在软件开发中使用工程原则,将其用于解决软件开发中的复杂问题。然而,绝大多数的现代软件能为用户提供好的服务,我们不应该让某些失败遮挡了过去数十年中软件工程师们取得的巨大成功。

另一个软件错误的例子③。

在 1991 年的海湾战争中,美国用船运送爱国者导弹到以色列,以保护以色列免受飞毛腿导弹的攻击。然而,一枚飞毛腿导弹穿过了爱国者反导弹防御措施,击中了沙特阿拉伯的

---

① F. P. Brooks Jr. No Silver Bullet. Essence and Acci-dents of Software Engineering [J]. Computer,1987,4

② Michael A. Cusumano. 微软的秘密. 北京:北京大学出版社,1997

③ S. R. Schach. 面向对象与传统软件工程. 第 5 版. 韩松,等译. 北京:北京机械工业出版社,2003

Dhahran 附近的一个兵营,一共有 28 名美国人死亡,98 人受伤。爱国者导弹的软件设计中有一个累积的定时错误,爱国者导弹每次只能工作几个小时,过了这个时间之后时钟就要复位。由于这个错误从未有过明显的影响,从而就没有被检测到。然而,在海湾战争中,爱国者导弹在 Dhahran 连续工作了 100 小时以上,这引起的累积时间误差大得足以导致系统的不精确。

以色列的军队仅在 8 个小时后就察觉到了这个定时问题,并立刻向美国的制造商报告了这个问题,制造商尽快地纠正了这个问题;然而悲惨的是,新软件在兵营被飞毛腿导弹直接击中的第二天才到达。

1.2　软件工程的定义及研究的内容

1.2.1　科学、工程与技术的界定

计算机科学、计算机工程、计算机技术等术语,人们几乎天天都会涉及到,但如果问这 3 个词语有何区别,即使是专业人员,也可能会陷入困境。所以在介绍软件工程之前,有必要对科学、工程与技术这 3 个基本术语做一解释,尽管这是很困难的。

"科学"(Science)是指探知事物的本质、特征、内在规律以及与其他事物的联系,是关于自然、社会和思维的发展与变化规律的知识体系。或者说,科学是建立在经验主义、实验以及方法论自然主义之上的各种知识。

随着人类文明的发展,人们可以建造出比单一产品更大、更复杂的产品,这些产品不再是结构或功能单一的东西,而是各种各样的所谓"人造系统"(如建筑物、轮船、飞机等),于是"工程"(Engineering)的概念就产生了,并且它逐渐发展为一门独立的学科。工程是指将自然科学原理应用到工农业等生产部门中而形成的各门学科的总称,如机械工程、水利工程、化学工程、系统工程等,而本书所讨论的是软件工程。

"技术"(Technology)则是运用科学规律解决实现某一目的的手段和方法,泛指根据生产实践经验和科学原理而发展形成的各种工艺操作方法、技能和技巧。前者是认识世界,后者则是改造世界。人们通常讲的"科学实验"实际已将科学与技术联系在一起,经过实验证明的正确理论是科学,否则就不是科学或是"伪科学"。科学与技术的密切结合成为变革世界的巨大动力。

事实上,软件的开发到底是一门科学还是一门工程,这是一个被争论了很久的问题。实际上,软件开发兼有两者的特点。但是这并不意味着它们可以被互相混淆。如果从计算科学的角度来看 3 者的关系,可表示为如图 1-6 所示的结构。

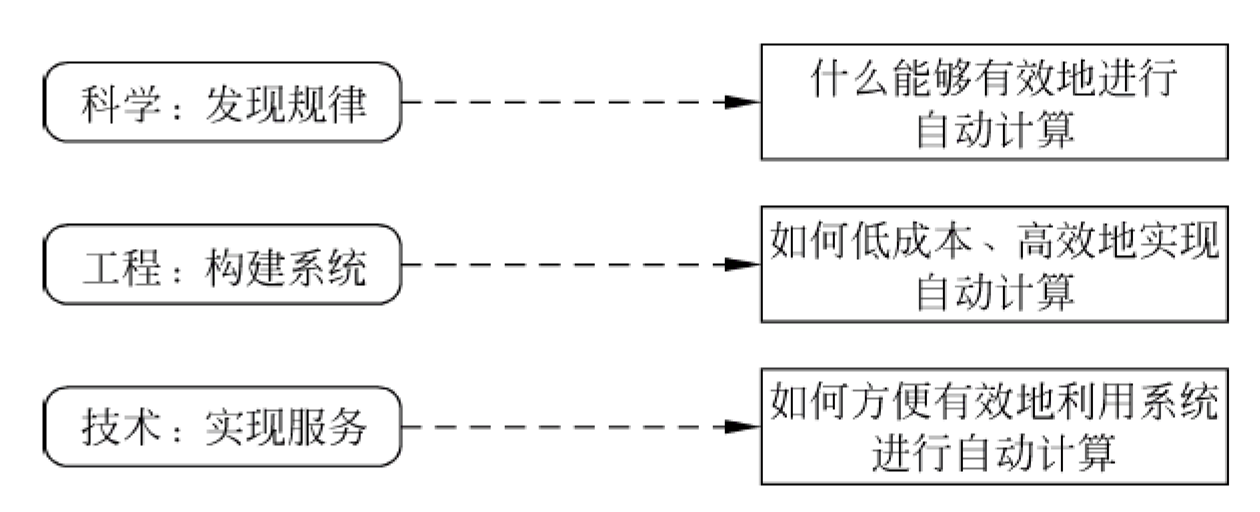


图 1-6　计算机科学领域下的科学、工程与技术

关于科学和技术的关系，有两种情况。一种是理论领先，带动技术进步，如先有电磁理论，后有它的应用。另一种是技术先行，然后才有对技术的概括和提炼，上升为理论，如激光技术在先，理论在后。在计算机科学技术领域，图灵等人的可计算理论和关于二进制的逻辑运算理论在先，计算机体系结构和运算技术在后。但是，从信息科学知识体系的总体上看，信息科学与信息技术的关系，两种情况皆有。因为，在某些领域信息技术发展在先，而它们的理论综合在后。

1.2.2 软件工程的定义与原理

1. 软件工程的定义

软件工程是一门工程学科，涉及软件生产的各个方面和整个过程。自从1968年提出软件工程这个术语，对于软件工程就有了各种各样的定义，其基本思想都是强调在软件开发过程中应用工程化原则的重要性。随着软件工程的不断发展和人们对软件和软件工程的深入认识，出现了对软件工程的各种各样的定义。

在首次NATO会议上Fritz Bauer给出的软件工程的定义是：软件工程是为了经济地获得可靠的和能在实际机器上高效运行的软件而确立和使用的一系列完善的工程原理（方法）。

著名的软件工程专家勃姆（B. W. Boehm，如图1-7所示）对软件工程的定义为：软件工程是现代科学技术知识在设计和构造计算机程序中的实际应用，其中包括管理在开发、运行和维护这些程序的过程中所必需的相关文档资料。

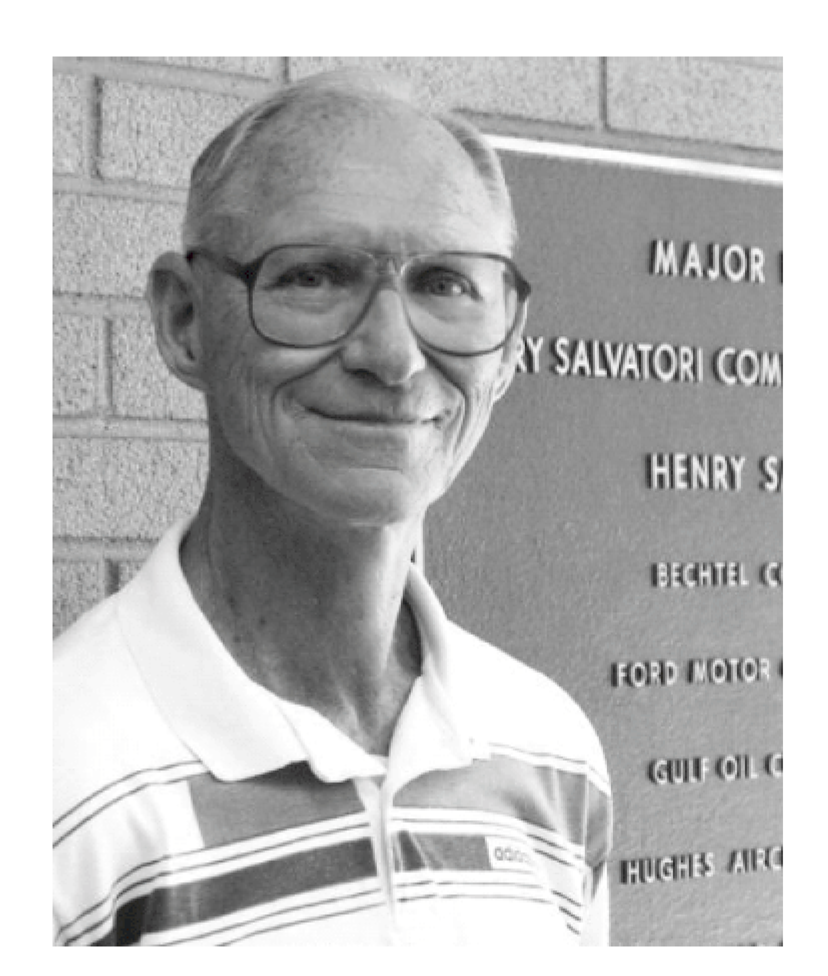


图 1-7　B. W. Boehm

1983年IEEE在其《IEEE软件工程标准术语》中对软件工程下的定义为：软件工程是开发、运行、维护和修复软件的系统方法。其中的“软件”是指计算机程序、方法、规则、相关的文档资料和程序运行所必需的数据。

1993年，IEEE给出了一个更加综合的定义：

① 将系统的、规范的、可量化的方法应用于软件的开发、运行和维护，即将工程化方法应用于软件；

② 在①中所述方法的研究[①]。

从以上定义可见，软件工程是一门指导软件开发的工程学科，它以计算机理论及其他相关学科的理论为指导，采用工程化的概念、原理、技术和方法进行软件的开发和维护。软件工程研究的目标是“以较少的投资获取高质量的软件”。

需要指出的是，软件工程和传统工程相比具有其特殊性。传统工程的学科基础只需依赖某些基本原理集和自然法则就能控制系统的行为并指导开发过程，而软件是知识产品，软件开发者的自由度较大，进度和质量都较难度量，生产效率也较难保证，并且软件系统的复

① IEEE Standards Collection：Software Engineering. IEEE Standards 610.12-1990，IEEE，1993

杂程度也是超乎想象的。因此软件仍然是在危机中生存和发展，生存源自于时代的需求，发展得益于人们的不懈努力。

2. 软件工程的原理

自从NATO会议上正式提出并使用“软件工程”这个术语以来，研究软件工程的专家学者们陆续提出了100多条关于软件工程的准则或“信条”。Boehm综合这些学者们的意见并总结了多年开发软件的经验，于1983年在一篇论文中提出了软件工程的7条基本原理①：

① 用分阶段的生命周期计划严格管理；
② 坚持进行阶段评审；
③ 实行严格的产品控制；
④ 采用现代程序设计技术；
⑤ 结果应能清楚地审查；
⑥ 开发小组的人员应该少而精；
⑦ 承认不断改进软件工程实践的必要性。

Boehm认为，这7条原理是确保软件产品质量和开发效率的原理的最小集合。它们是相互独立的，是缺一不可的最小集合；同时，它们又是相当完备的。它们可以任意组合蕴涵或派生，不仅适合以前，而且也适合现在和将来。

1.2.3　软件工程的3个要素

“软件就是程序，开发软件就是编写程序”这种错误观点的长期存在，影响了软件工程的正常发展。那么单纯的编程和软件工程之间有什么不同呢？区别就如同在院子里做一张桌子和在河上建造一座大桥之间的区别。这种区别主要表现在项目的数量级及所需的专业知识上。与做一张桌子不同，建造大桥需要大量的专业技能和高度的社会责任感，这就是严格的需求分析和量化的质量标准②。

软件工程是一门新兴的边缘学科，涉及的学科多，研究的范围广。归结起来软件工程研究的主要内容有方法、工具和过程3个要素，它们构成了一种层次化的技术③，软件工程层次图如图1-8所示。整个体系结构反映了以质量为中心的观点。关注质量是软件工程的根本出发点和最终目标。



图1-8　软件工程层次图

软件工程方法包括管理方法和技术方法，提供如何完成过程活动的指南和准则。如管理方面的软件重用技术、项目管理等，技术方面的面向对象分析、设计、实现技术、测试技术等。结构化方法和面向对象方法在软件开发方法中产生了较大的影响。

---

① 张海藩. 软件工程导论. 第3版. 北京：清华大学出版社，1998
② ERIC J. BRAUDE. 软件工程——面向对象的视角. 和华，刘海燕，等译. 北京：电子工业出版社，2004：20
③ Roger S. Pressman. 软件工程——实践者的研究方法. 第6版. 郑人杰，等译. 北京：机械工业出版社

软件工具为软件工程方法提供支持，研究支撑软件开发方法的工具，建立软件工程环境，为方法的运用提供自动或者半自动的支撑环境，软件工具的集成环境，又称为计算机辅助软件工程(CASE)。

软件过程则是指将软件工程方法与软件工具相结合，实现合理、及时地进行软件开发的目的，为开发高质量软件规定各项任务的工作步骤。软件工程的根基在于质量关注点(Quality Focus)。

需要强调的是，随着人们对软件系统研究的逐渐深入，软件工程所研究的内容也不是一成不变的。软件工程是在软件生产中采用工程化的方法，这种工程化的思想贯穿软件开发和维护的全过程。

1.2.4 软件开发方法——对客观世界的认知观

软件方法学是从各种不同角度、不同思路去认识软件的本质。整个软件的发展历程使人们越来越认识到应按客观世界规律去解决软件方法学问题。

1. 软件开发方法是一种认知观

毛泽东曾在 20 世纪 70 年代提出“三个世界”划分的战略思想，又称“三个世界理论”，是对当时国内国外形势做出的客观估计。软件的实质也主要牵涉“现实世界—概念世界—计算机世界”3 个空间。因而软件设计方法也应在这 3 个范畴的极限之内寻求发展，特别是“现实世界”。因为软件的实质是人们以计算机编程语言为桥梁，将现实世界映射于计算机世界中，以解决人们在客观感知世界中的问题(如图 1-9 所示)。



图 1-9 从现实世界到计算机世界的认知过程

一个很有趣的事实是，在很多工程领域中，从事工程设计与开发的都是熟悉本领域的工程师或专家。而在软件工程领域中，一般是由具有计算机专业背景的人在另一个并不熟悉的领域中从事软件开发，为具有另一种文化背景的人创造产品，这个特性与前面介绍的软件特性紧密相关。例如，软件工程师是诸如 Java 程序设计、软件体系结构、测试或统一建模语言(UML)等方面的专家，他们通常并不是图书馆管理、工业控制或银行事务等领域的专家，但是他们却不得不为这些领域开发应用系统。缺乏应用领域的相关知识，是软件开发项目出现问题的常见原因。软件工程师不仅缺乏应用领域的实际知识，他们还缺乏该领域的文化知识。例如，软件开发者通过访谈、阅读书面文件等方法了解到用户组织的“正式”工作流程，然后用软件实现了这个工作流程。但是，决定软件系统成功与否的关键问题是，用户组织是否真正遵守这个工作流程。对于局外人来说，这个问题更难回答。为此，软件工程师们

只能利用一些定义好的技术集及符号来表示业务处理过程，一般表示成一系列的步骤，每一步骤都与相应的技术和符号相关，即利用软件分析与设计方法建模，并以此来组织软件生产。在这里，对应用领域的认识观在软件开发中起到关键作用，不同的认识观决定了不同的开发方法和软件产品质量。

为了解决以上问题，经过30多年的研究与实践，人们已经摸索和建立了多种软件工程方法。例如，结构化方法、形式化方法、面向对象方法、基于构件的方法、基于Agent的方法、基于净室技术的方法、基于敏捷技术的方法等。这些方法在自身的发展过程中又不断吸收其他方法和技术的长处，导致新技术新方法层出不穷，成为现代软件工程发展过程中的亮点，从而不断丰富和发展了软件工程的理论与实践。

下面简单介绍在软件开发方法中常采用的结构化开发方法和面向对象方法，这两种方法也是本书在第2篇和第3篇中要介绍的内容。

2. 面向过程的结构化开发方法

1969年，E. W. Dijkstra首先提出了结构化程序设计的概念。他的认知观认为："人的智力是有限的"，软件开发是一项复杂的工程，它强调了从程序结构和风格上来研究程序设计，由此称为结构程序设计方法。这方面的重要成果就是在20世纪70年代风靡一时的结构化开发方法，即面向过程的开发或结构化方法的产生。

结构化开发方法由结构化分析、结构化设计和结构化程序设计三部分有机组合而成。这里所说的结构是指软件系统内各个组成要素之间的相互联系、相互作用的框架。结构化分析方法给出一组帮助系统分析人员产生功能规约的原理与技术。它一般利用图形表达用户需求。结构化设计方法给出一组帮助设计人员在模块层次上区分设计质量的原理与技术。它通常与结构化分析方法衔接起来使用。结构化程序设计方法遵循以模块功能和处理过程设计的基本原则，追求过程化、模块化、封装以及更高的抽象的结果，使一个庞大复杂的问题在结构上得以简化从而得到一个结构清晰的程序。

结构化方法经过近40年的发展，已经形成了一套比较成熟的理论。结构化分析使用需求建模方法，以数据流图和控制流图为基础，由系统分析员划分出流变换函数，以得到系统的软件结构，并将其映射为软件功能。其次用状态迁移图来创建行为模型，用数据词典开发成数据模型。

在本质上，结构化的软件开发方法是以面向数据、面向过程、面向功能、面向数据流的观点来映射问题的。设计关注的是如何用函数和过程来实现对现实世界的模拟，将其映射到计算机世界之中。在此基础上再借助某种形式语言，抽象出变量、表达式、运算、语句等概念。但在这个层面上，对客观问题的有效认知方法还没有根本得到解决。

3. 面向对象的开发方法

Wittgenstein是21世纪乃至人类哲学史上最伟大的哲学家之一。他生前于1922年出版了一本著作——《逻辑哲学论》(《Tractates Logico-Philosophicus》)。在该书中，他阐述了一种世界观，或者说一种认识世界的观点，这种观点，在八九十年后的今天，终于由一种哲学思想沉淀到技术的层面上来，成为计算机软件开发方法的主流，这就是OO(Object-

Oriented,面向对象)。他提出,对象是简单的(基本的),对象形成世界的实体,因而它们不会是复合物。

根据对 Wittgenstein“世界—事实—原子事实—对象”的 OO 思想分析,它是一个从整体到局部、从抽象到具体的认识之链。其中,对象作为最基本的模块,是整个认知中的基本元素,对象通过相互之间的复杂关联构成了整个世界。实际上这个观点也是面向对象理论的根本要素。

面向对象方法学认为,客观世界是由各种对象组成的,任何事物都是对象,复杂的对象可以由比较简单的对象以某种方式组合起来。因此,面向对象的软件系统是由对象组成的,对象是软件模块化的一种新的单位,它代替了基于功能分解方法中的所谓“模块”等传统的观点。对象将数据和过程封装在一起,这同传统的方法中将数据和过程分别对待和处理形成了鲜明的对比。面向对象技术比较适于大型软件系统的开发。近年来在许多应用领域中面向对象方法学已经迅速取代了传统的方法学。

面向对象方法学有 4 个要点,可以用以下公式概括(如图 1-10 所示):

面向对象方法＝对象＋类＋继承＋消息



图 1-10　面向对象方法学

面向对象方法就是既使用对象又使用类和继承等机制,而且对象之间仅能通过传递消息实现彼此通信的方法。除此之外,采用该技术开发的软件工程项目还具有稳定性好、可重用性高、可维护性强等优点。

20 世纪 80 年代末以来,出现了如 Coad/Yourdon、Booch、OMT、Jackson 等几十种面向对象方法。尤其值得一提的是统一建模语言(Unified Modeling Language,UML)的出现,它从多种方法和工程实践中吸收了许多经过实际检验的概念和技术,统一了符号体系,给软件工程带来了新气象,是软件工程发展过程中一个具有里程碑性质的新进展。

从以上的讨论可知,软件工程的发展历史就是对客观世界的认知观不断发展的历史,软件开发方法就是人们不断追求更高的抽象,更加符合人们认知规律的过程。面向对象的方法当然不会是历史的终结。随着网络技术、开发平台技术、中间件技术和人工智能技术的发展,有力地带动了计算机软件、硬件的更新和升级,给软件工程方法的研究也带来了更多的机遇和发展空间。而每前进一步,人们对客观世界的认识观就有了新的飞跃,看到了未来的探索方向。

1.2.5　软件工程与相关科学的关系

软件工程的概念最初仅仅是将传统的工程原则应用于解决软件开发的问题,但是随着计算技术在解决复杂问题方面的广泛应用,计算机科学已经成为一门基础学科,软件工程的学科范畴也逐渐清晰。软件工程学科的理论基础是数学和计算机科学。它又是一门交叉性的工程学科,软件工程的相关学科有计算机科学、数学、工程科学、管理学等。计算机科学和数学用于构造软件的模型与算法;工程科学用于制定规范、分析建模、结构设计等;管理科学用于项目计划、资源、质量、成本等管理。软件工程的重点在于大型软件的分析与评价、规

格说明、设计和演化，同时涉及管理、质量、创新、标准、个人技能、团队协作和专业实践等。此外，软件工程还十分重视管理过程，以提高软件产品的质量、降低开发成本、保证工程按时完成。系统性、规范性、可度量性也是软件工程非常关注的。软件工程与其他学科的关系如图 1-11 所示。

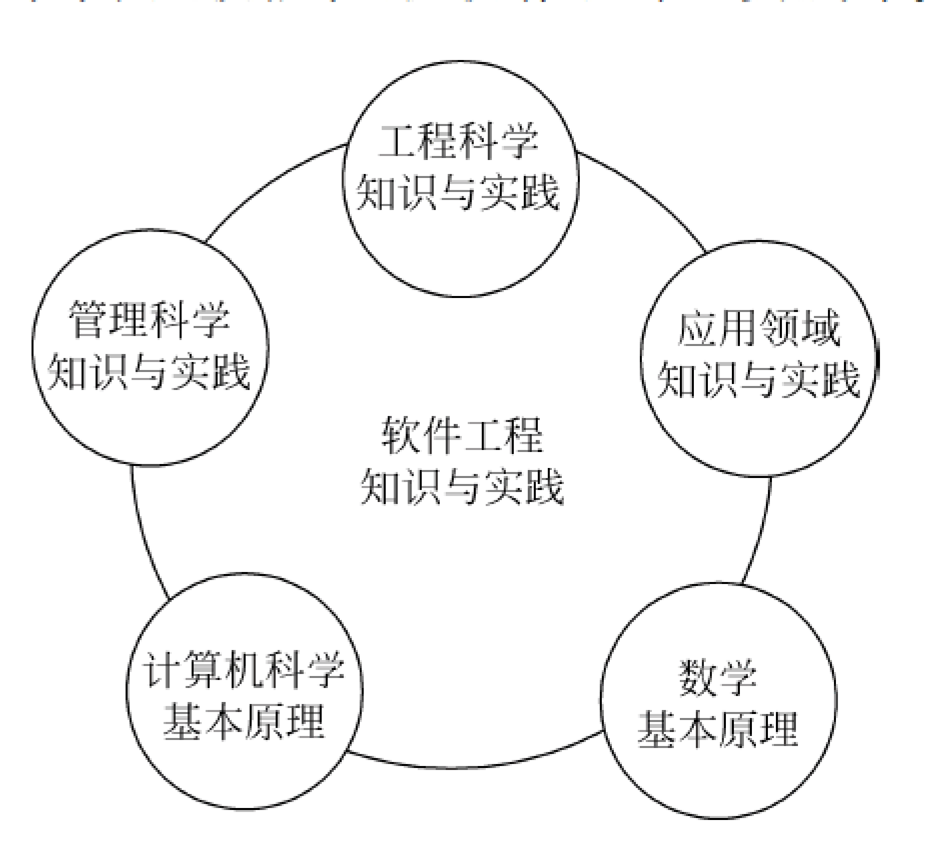


图 1-11　软件工程与其他学科的关系

软件工程的研究和实践涉及人力、技术、资金、进度的综合管理，是开展最优化生产活动的过程；软件工程必须划分系统的边界，给出系统的解决方案。因此，工程活动以设计为中心，设计在软件工程活动中占有十分重要的地位。为了满足项目需求，工程设计过程必须对潜在的冲突和约束进行折中。工程设计涉及技术、经济、法律和社会等方面的问题。因为软件的特殊性，软件工程与传统的工程学不同。软件工程更关注抽象、建模、信息组织和表示、变更管理等。软件工程在产品的设计阶段必须考虑实现和质量控制。持续的演化是软件产品的重要特征。软件工程设计的关键是工程设计决策，它将用于软件抽象的各个层次。重用和基于构件开发在工程设计实践中越来越受到重视。

软件开发是一个项目目标实现的过程，管理科学的目标性和约束性原则在软件工程中得到重要的体现。软件工程强调软件产品及其开发过程的成本、进度、质量和文档的属性，要求在特定的环境和一定的组织机构内，有效地利用有限资源（人力、物力、财力等），通过协调一系列相互关联的任务，在规定的时间内完成，并满足一定的性能、质量、数量、技术指标等要求。由于软件的特殊性，增大了管理的难度。因此，软件工程在软件生存周期的整个过程中，对需求、计划、成本、风险、过程和质量进行度量、跟踪、管理与控制。

软件专业的学生必须掌握该学科的基本技能和知识，具备理论与实践相结合的能力，认识抽象和建模的重要性，同时还需要了解与学习计算学科以外的领域知识，做到能够支持该领域的软件开发，并充分意识到优秀设计的价值。

1.3　软件工程的教育与知识体系

1.3.1　软件工程的教育体系

随着世界经济的多元化发展，以计算机为基础的信息技术迅速扩展到各个领域，社会和人类对信息的依赖迅速增长，计算机技术和基于计算机的应用技术已经成为信息社会的重要基础设施，计算机教育和培训也成为我国高等教育中一个重要的环节。近年来，行业界和教育界都再一次关注到“计算”（Computing）这个词的含义，并明显意识到计算所覆盖的领域在不断地、迅速地扩展。高等教育中学科的培养目标、教学计划和课程设置也随着领域的变化在不断地调整、巩固和完善。

IEEE/ACM[①] 一直在跟踪工业界对计算领域人才需求和教育界对人才教育培训的需求、状况、发展和存在的问题，并于 2001 年给出了具有指导性意义的计算学科本科教学参考计划 Computing Curriculum 2001。这个计划对我国计算机学科的教育产生了很大的影响，国内专家学者对其进行了详细研究，并于 2002 年公布了中国计算机本科教学参考计划(CCC2002)，在国内外也产生了很大的影响。继 CC2001 推出后，经过几年的跟踪研究、意见反馈和计划评估，IEEE/ACM 在总结前期工作的基础上，对原“计算教程 CC2001”给出的 4 个专业方向进行了修改和扩充，并给出了新的评述，于 2004 年 6 月公布，并把它们称为“计算教程 CC2004”(Computing Curriculum 2004)。宣布“信息技术已经加入计算学科的家族”，声称该文件的主要目的是让学术界了解计算学科包括哪些主要领域，如何比较它们各自的特点。

计算学科的分化表现了一种科学发展和知识演化与时俱进的趋势，IEEE/ACM 在解释这种分化情况时，对“计算”这个词的含义和它所覆盖的领域进行了如下精彩的刻画：“计算的概念在过去的 10 年里发生了巨大的变化，这种变化对教学计划的设计和教育方法会有深刻的影响。我们称之为‘计算’的概念已经拓展到难以用一个学科来定义的境地。我们过去形成的课程设置报告曾经是试图将计算机科学、计算机工程和软件工程融合成关于计算教育的一个统一的文件。这种做法在十年前也许是合理的，但我们确信 21 世纪的计算蕴涵有多个富有生命力的学科，它们分别有着自己的完整性和教育学特色。”

计算学科长期以来被认为代表了两个重要的领域，一个是计算机科学，另一个是计算机工程，两者曾经分别作为软件和硬件领域的代名词。随着科学技术的发展，IEEE/ACM 在 CC2001 中将计算学科分为 4 个领域，分别是计算机科学、计算机工程、软件工程和信息系统。近期的 CC2004 报告，在原来 4 个学科领域的基础上，增加了一个信息技术专业学科领域，并预留了未来的新发展领域。各个专业都针对本科生的教育，提出了相应的知识领域、知识单元和知识点，并给出了相应的参考教学计划和课程设置。

2001 年底，教育部开始推动示范性软件学院项目，正式在我国启动了软件工程学科的教学；2004 年，教育部计算机教学指导委员会又提出了计算机科学与技术专业分流培养方案，在参考 CC2001 的基础上，给出了中国自己的计算机专业本科教学参考计划，计算机学科原有的专业设置框架被突破，逐渐形成了在“计算机科学与技术”一个专业之下分为计算机科学(Computer Science，CS)、计算机工程(Computer Engineering，CE)、软件工程(Software Engineering，SE)、信息技术(Information Technology，IT)4 个专业方向的新格局[②]，并于 2005 年发布了 4 个方向的专业规范。

总的来看，目前有关“计算”的内涵和外延都在迅速地扩大，各个分支已经形成丰富和完整的知识体系，已经不可能将如此丰富的内容安排在一个单一的本科教学课程体系之中，而

---

① IEEE(Institute of Electrical and Electronics Engineers)是一个国际性的电子技术与信息科学工程师的协会。是美国规模最大的专业学会。IEEE 是一个非赢利性科技学会，拥有全球近 175 个国家 36 万多名会员。透过多元化的会员，该组织在太空、计算机、电信、生物医学、电力及消费性电子产品等领域中都是主要的权威。

美国计算机协会(Association of Computing Machinery，ACM)是一个世界性的计算机从业人员专业组织，创立于 1947 年，是世界上第一个科学性及教育性计算机学会。

② 教育部计算机科学与技术专业教学指导分委员会. 中国计算机本科专业发展战略研究报告暨专业规范. 北京：高等教育出版社，2006，9

且单一的培养模式也不能满足社会对多种规格人才的需求，必须制定和实施不同的培养计划才能满足这种不同的需要。

1.3.2　CC2005 的 4 个方向专业规范

作为在计算学科教育方面最有代表性和影响力的工作，在 CC2005(Overview Report)中，IEEE/ACM 为计算学科的知识空间给出了一个二维图解。4 个专业方向的知识体系以明显不同的位置和区域体现在这个空间里，如图 1-12 所示。该空间的横坐标从左至右，表示知识的性质从理论到应用，纵坐标从下至上，表示计算系统知识的层次从内向外，从硬件到软件，最后到组织机构的行为。

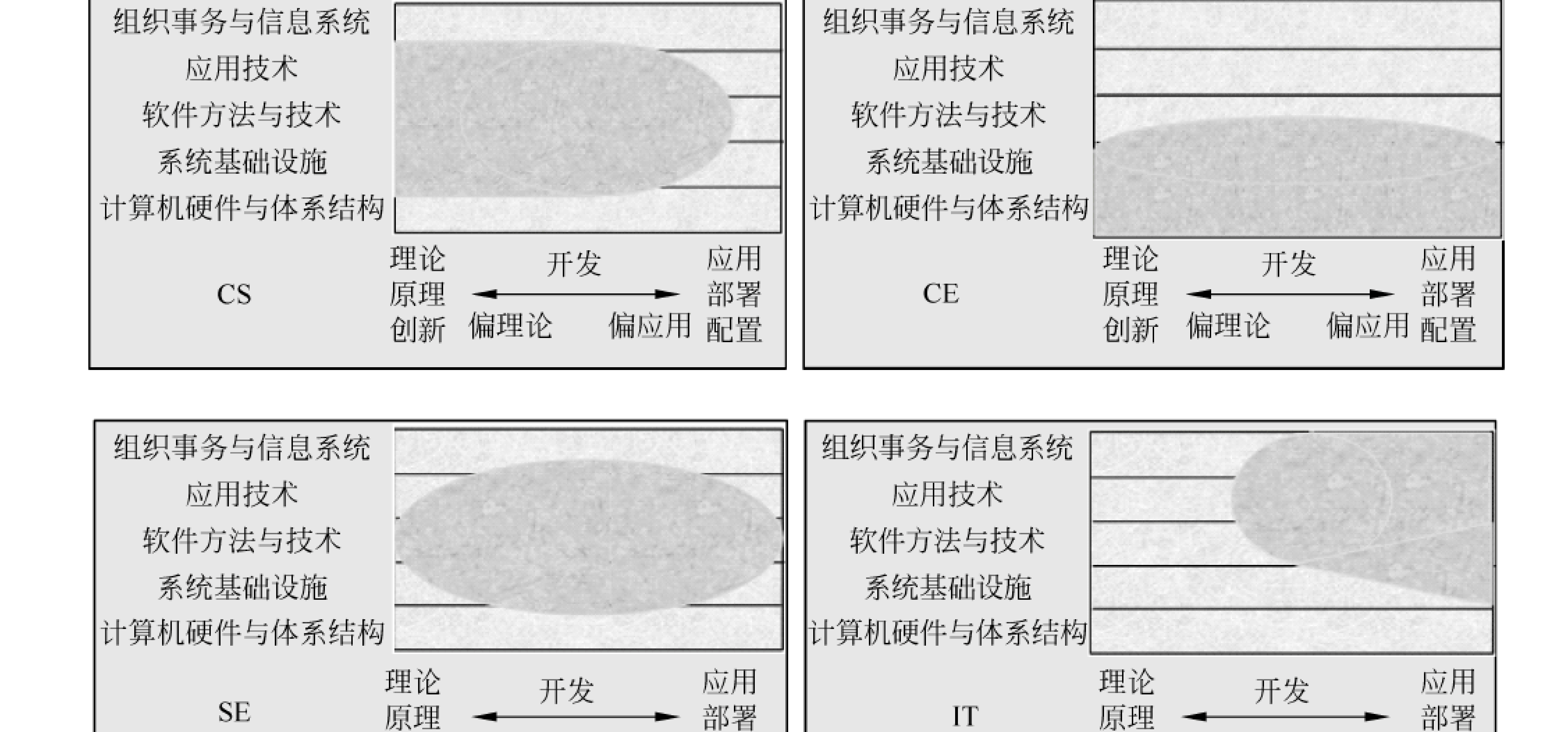


图 1-12　CC2005 给出的计算学科知识空间的二维图解

CC2004 报告强调了对软件工程的新定义，即软件工程是"以系统的、学科的、定量的途径，把工程应用于软件的开发、运营和维护；同时，开展对上述过程中各种方法和途径的研究"。这里明确提出了"把工程应用于软件"，明显地体现了软件工程领域内的两类重要的研究和应用方向：工程学和方法学。

软件工程专业教育培养的目标是让受教育者了解和掌握软件开发中的方法学和工程学的知识，并应用于实践。因此，IEEE 区分了典型的人才职业特征，这些特征也强调了其职业定位(这里的"职业"体现了专业技术背景和应用行业背景)。例如，电子工程师着重关注电路、信号分析、信号合成、信号传输等电子学、物理学领域的问题；计算机科学家主要关注计算的理论基础和算法；计算机工程师着重关注基于计算机的产品的正常运行和维护；信息资源专家则关注信息资源的获取、部署、管理及使用；而软件工程师则重点关注大规模软件开发与维护的原则，着重开发过程质量，以避免潜在的风险性。

针对 CC2004 报告，IEEE/ACM 软件工程学科组于 2004 年 5 月 21 日公布了软件工程教育知识体系(Software Engineering Education Knowledge，SEEK)的最终报告，这份报告

针对软件工程本科教育的课程知识领域，给出了相关的领域方向的课程知识单元和知识点的配置，以及参考课程计划。

1.3.3 软件工程的知识体系——SWEBOK

2001年5月，国际标准化组织（ISO）和国际电工委员会（IEC）共同发布了关于软件工程的一份特殊的标准化文件——《软件工程知识体系指南》，启动了为期两年的指导软件工程学科教育和职业培训的试用过程。2004年6月23日，IEEE的另一个学科组也公布了软件工程知识体系（Software Engineering Body of Knowledge，SWEBOK）的更新版，它被软件行业称为软件工程教育的基本法。这两个知识体系分别面向本科软件工程教育和软件工程行业教育和从业要求，“用国际标准支持软件工程教育”的口号在国际上提了出来。国际上许多涉及计算机科学的高等学府、职业培训机构和有关的政府机构为之纷纷予以响应。标准化与学科教育和职业培训本身也走到了一起。

1.《软件工程知识体系指南》的产生背景与意义

《软件工程知识体系指南》为什么在如此短的时间就受到国际青睐？这是由其产业背景和教育需求所决定的。软件工程职业化是软件工程成熟的标志，正如化学工程独立于化学，航空工程独立于航空学一样，软件工程是一门独立的学科，应该有自己的职业体系和教育课程体系。随着软件产业的逐渐形成，一方面，国际软件工程标准化迅速活跃起来，另一方面，软件工程的教育也应运而生。特别是20世纪80年代和90年代，计算机科学教育得到突飞猛进的发展，进一步带动了软件工程教育。不过，人们发现，虽然许多院校的大纲已经从最初的以程序设计语言和编码为中心的课程设置转移到强调软件工程理论和设计，但是，直接面向“工程化”的课程和学时很少。然而，恰恰是诸如需求建模、设计方法、体系结构设计、软件复用、软件过程、质量问题、团队组织技能之类的软件工程领域的知识和技能对于商业软件的高效开发是至关重要的。由于缺乏对于各种软件工程化实践活动和必要能力的共识，致使软件工程化活动出现许多混乱现象，对软件工程知识的评价、获取和应用造成严重不良后果。当软件工程化已经被人们广泛接受并且成为关注对象时，对于权威的关于软件工程知识结构的描述的需求也就成为必然[①]。

高等院校各自按照自己的人才培养方案实施着各自认为合适的教学大纲，培训机构按照自己认为适宜的内容选择或编写教材，研究者们发表着各执一词的关于软件工程知识的论述，如此种种无不迸发出统一软件工程知识结构需求的呼声。美国最先感受到对软件工程教育的这种迫切需求。美国计算机协会（ACM）、电子电气工程师学会计算机学会（IEEE-CS）和计算机科学认可委员会（CSAB）等纷纷极力倡导和支持开发高质量的软件工程课程并且积极提供指导。从1993年起，IEEE-CS和ACM一直积极促进软件工程的专业化。20世纪末期，一项向美国联邦政府提出的关于软件工程体系知识需求的项目建议，把犹如沧海横流的对于软件工程教育的需求汇流于一脉。

1999年5月，ISO和IEC的第一联合技术委员会（ISO/IEC JTC 1）为顺应这种需求，立

① 吴源俊. 软件工程知识结构[J]. 软件工程与标准化，2002，5：32～41

即启动了标准化项目——“软件工程知识体系指南”(Guide to the Software Engineering Body of Knowledge),简称 SWEBOK 指南。

经过多年的努力,2004 年 6 月,美国 IEEE 协会和 ACM 的联合网站上公布了软件工程知识体系(SWEBOK)2004 版全文,这标志着 SWEBOK 项目的工作告一段落,软件工程作为一门学科,为取得对其核心的知识体系的共识,已经达到了一个重要的里程碑。SWEBOK 的初步成就,无疑对我国亟待快速发展的软件产业界软件工程教育调整了方向盘,增添了助推器,因而引起业界的广泛重视。SWEBOK 是指南,不是软件工程知识本身,软件工程方面的知识已经存在。SWEBOK 的宗旨是在众多知识中圈定“属于”软件工程学科领域的知识,规定软件工程教育和培训方面的要求和制定出评价原则,并且在世界范围内求得公认。它还试图为从事软件实践的工程师和负责制定有关职业指导原则的政策的官员提供指南。此外,制定高等院校课程认证规则、认可政策和职业实践指南的社会职业工作者和教育工作者以及从事软件工程职业学习的学生也可以从 SWEBOK 指南中得到收益。

2. SWEBOK 的目的与内容

SWEBOK 指南开宗明义提出 5 个目的[①]:

(1) 促进软件工程业界统一看法。

(2) 划定学科边界,澄清软件工程的学科地位。

(3) 刻画软件工程的学科内容。

(4) 提出访问 SWEBOK 的论题(知识点)。

(5) 为个人认证、申请执照、课程体系制定提供基础。

SWEBOK 把整个体系分解为 10 个知识域(Knowledge Area)[②]:软件需求(Software Requirements)、软件设计(Software Design)、软件构造(Software Construction)、软件测试(Software Testing)、软件维护 (Software Maintenance)、软件配置管理 (Software Configuration Management)、软件工程管理 (Software Engineering Management)、软件工程过程 (Software Engineering Process)、软件工程工具与方法(Software Engineering Tools and Methods)和软件质量(Software Processes and Product Quality),其知识域结构如图 1-13 所示。

SWEBOK 的每个知识域又分为若干子域,每个子域分为若干论题(Topic),我国学界称之为知识点,每个知识点还可以再分为下层,或下下层的子知识点。SWEBOK 只给出知识域确切的概念和准确的定义,即内涵定义。从知识域到子域到知识点,要完全理解知识域的含义还要靠它的外延,即各种参考文献。

SWEBOK 指南将每个知识域作为一章详细描述,加上导言(第 1 章)和相关学科(第 12 章)共 12 章,此外,还包括 4 个附录 A、B、C、D。指南全文文本就是对第 5 个目的的支持。有关详细内容请参考 SWEBOK 网络资源:http://www.swebok.org/pub2.html。

---

① 麦中凡. 解读-SWEBOK 2004[J]. 计算机教育,2004.10

② http://www.swebok.org/pdfformat.html

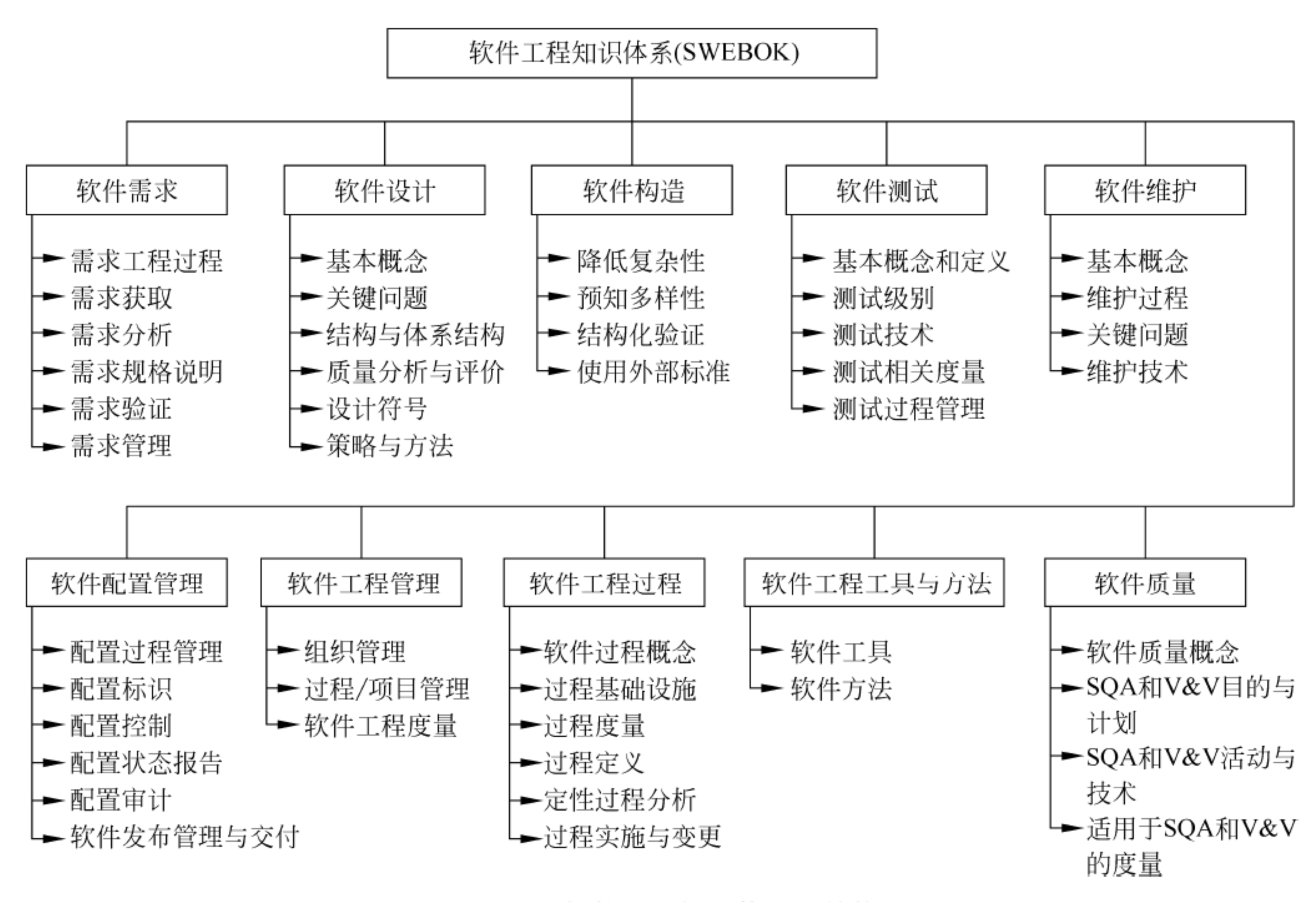


图 1-13　软件工程知识体系的结构

1.4　软件工程的标准

标准是一个“在经常和反复的使用中构成了活动或其结果的规则、原则或特征，并由共识确立或者公认机构批准的文件，其目的是在既定的环境中实现最佳程度的秩序”①。每一个应用领域一般都有一套公认的经常以规章制度形式颁布的标准和做法。软件工程也不例外。为适应软件产品的需要，国际标准化组织和一些标准化机构深入开展了软件标准化工作，并初步形成了一套标准化体系。

1.4.1　软件工程标准化的意义

人们的社会生活离不开交往。在交往中最先遇到和首先要解决的是通信工具——语言文字问题，标准化是组织现代化生产和工程建设的重要手段，是进行科学质量管理的重要组成部分。软件项目的开发是一项复杂的系统工程，要运用系统的思想和系统工程的方法，采用工程化方法和工程途径来研制与维护软件；采用先进的技术、方法与工具来开发与设计软件；以工程化的理念来管理和规范软件。

软件标准化是指软件产品的功能、开发过程和质量保证体系的标准化②。其形成过程是通过对软件工程过程乃至软件生产的技术、方法、工具和管理活动等进行深入的研究后，

① 项目管理协会. 项目管理知识体系指南. 第 3 版

② 软件标准化与质量工作组. 我国软件标准化的发展战略与对策[J]. 中国标准化，1997(9)

将其技术、方法、工具、经验等总结成条例，并通过这些条例，为软件分析、设计、实施、运行、维护以及软件工程的过程控制提供指南，为软件产品的质量保证和软件工程管理提供导则，为软件工程技术、方法和工具的使用提供约束和技术指导。标准化是软件产业健康发展的强力支撑，也是软件工程成熟的重要标志。

软件工程标准化的意义是显而易见的。由于软件质量的控制具有其特殊的复杂性，标准可以对软件系统提出质量需求，指导软件的开发、测试和维护，从而获得高质量的软件和质量测定结果。仅就一个软件开发项目来说，有多个层次、不同分工的人员相配合，在开发项目的各个部分以及各开发阶段之间也都存在着许多联系和衔接问题。如何把这些错综复杂的关系协调好，需要有一系列统一的约束和规定。在软件开发项目取得阶段成果或最后完成时，需要进行阶段评审和验收测试。投入运行的软件，其维护工作中遇到的问题又与开发工作有着密切的关系。软件的管理工作则渗透到软件生存期的每一个环节。所有这些都要求提供统一的行动规范和衡量准则，使各项工作都能有章可循。

软件工程的标准化会给软件工作带来许多好处，例如，提高软件的可靠性、可维护性和可移植性(这表明软件工程标准化可提高软件产品的质量)；提高软件的生产率；提高软件人员的技术水平；提高软件人员之间的通信效率，减少差错和误解；有利于软件管理；有利于降低软件产品的成本和运行维护成本；有利于缩短软件开发周期。

1.4.2　软件工程的国际标准与体系

1. 发展概况

为了克服这种“软件危机”，西方发达国家在20世纪70年代初就意识到建立软件标准的重要性，并着手研究和制定各种软件工程标准。美国国防部为了适应军用软件的开发需要，制定了一系列软件工程标准，如美国国防部标准DOD-STD-1679A和DOD-STD-2167，它们构成了一个体系，比民用标准更加严格具体。目前世界各国都在努力研究新的软件开发方法，制定软件工业化生产的标准与规范，并进一步完善软件工程环境。

国际上从事软件工程标准化的机构主要有：国际标准化组织和国际电工技术标准化委员会第1联合技术委员会第7分委员会(ISO/IEC/TC1/SC7)“软件工程”、国际标准化组织第176技术委员会(ISO/TC176)“质量管理和质量保证”和美国电工电子工程师学会(IEEE)。

随着软件工程标准化的发展，目前制定的标准不规定开发模型，只定义了一些开发过程和活动，使用者可根据自己的开发模型来剪裁这些过程和活动，这样增加了灵活性，扩大了使用范围。典型的标准有ISO/IEC 12207和IEEE 1074。另一方面，现行的标准特别注重软件企业、软件开发活动的管理，尤其是质量管理和质量评定，典型的标准有ISO 9000系列标准、软件能力成熟度模型以及ISO/IEC 14598、ISO/IEC TR15504系列标准。

目前，国际上已形成了较完善的软件工程标准体系，如图1-14所示[①]。



图1-14　国际软件工程标准体系

① 冯惠. 软件工程标准化[J]. 中国标准化，2002

随着软件和网络技术的发展,国际上的标准化机构又提出许多软件标准化新课题,主要有网络通信和网络计算机(包括 WWW 浏览器、Web 检索工具、防火墙等)、多媒体(涉及面向客户服务器媒体技术)、"中间层媒体"(也称中件或媒体,包括软构件技术和嵌入式芯片)等。

2. 国际标准化组织

国际标准化组织(ISO)非常重视软件工程标准化,1982 年专门成立了一个分技术委员会负责这项工作,即 ISO/TC97/SC7(现在的 ISO/ IEC JTC1/SC7)。1987 年 ISO 发布了国际质量保证 ISO 9000 标准系列,其中 ISO 9000-3 就是专门针对软件行业制定的关于软件质量管理和质量保证的国际标准。在软件工程方面,ISO 已颁布了一系列关于软件开发、测试、质量度量、用户文档编制和开发规范等方面的国际标准,其中有些是等同采用 IEEE 的标准。

目前,该组织共发布各类软件工程标准 80 多部,其标准体系如图 1-15 所示[①]。

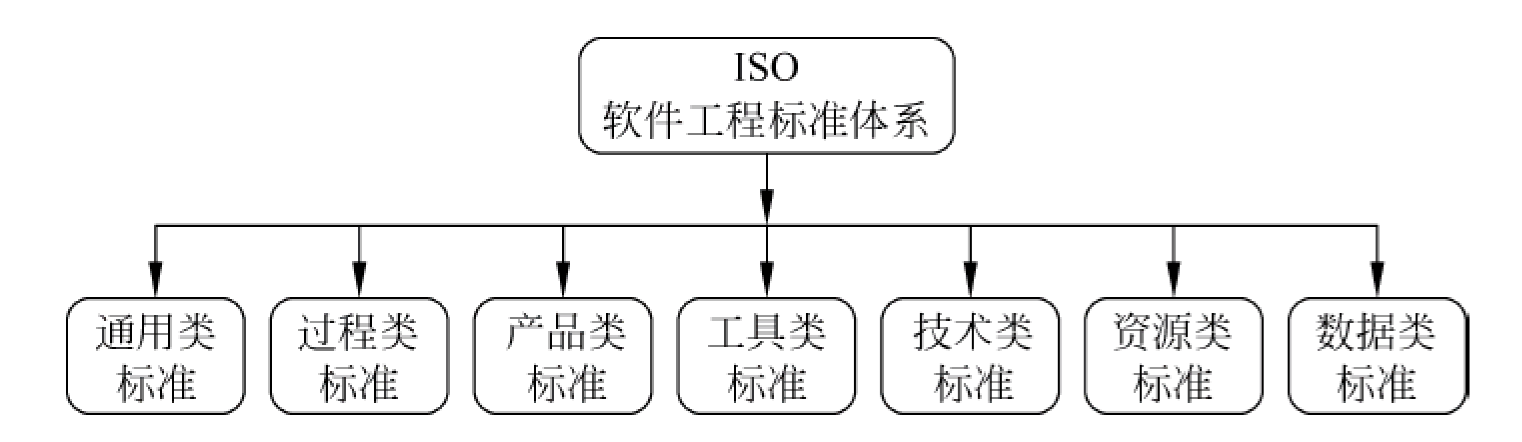


图 1-15 ISO 的软件工程标准体系

3. IEEE

IEEE 是最早开展软件工程标准研究制定的组织,它制定的许多标准被直接采纳为美国国家标准,并且许多国际标准是以 IEEE 标准为基础制定的,例如,软件生存周期过程、风险管理、软件重用过程等,目前已制定了 50 多项软件工程标准。

在软件工程标准方面,IEEE 更贴近于软件工程的实际,如《IEEE 软件质量保证计划》、《IEEE 软件配置管理计划》、《IEEE 软件测试文档标准》、《IEEE 软件需求规格说明的实施建议》、《IEEE 软件单元测试标准》、《IEEE 软件验证和确认标准》、《IEEE 软件用户文档标准》等都具有非常高的实用性。

现在,IEEE 软件工程标准的制定工作沿着以下两个主要方向进行。

(1) 为现有标准提供补充培训材料。

(2) 扩充产品标准的相关过程标准和度量方法标准。

为保持软件工程标准之间的协调一致,IEEE 每 5 年修订或确认一次。

1.4.3 国家标准

我国的软件工程标准化起步于 1984 年。同年,全国信息技术标准化技术委员会的前身——全国计算机与信息处理标准化技术委员会成立了软件工程分技术委员会。在委员会

① 李刚.软件工程标准化现状与分析[J].四川大学学报(工程科学版),2007,39:73~76

卓有成效的组织和参与下，到目前为止共制定国家标准和行业标准 20 多项，这些标准主要是采用国际标准和 IEEE 标准而制定的。例如，《软件过程能力评估模型》和《软件能力成熟度模型》两项行业标准，给出了软件工程活动的总体框架和评估准则。为企业软件工程管理水平跃升指明了方向。这对处于手工作坊式的软件企业走上工程化的道路以及顺应企业的发展潮流，无疑是一次强有力的推动和促进。

从目前我国标准的制定情况来看，通过引入国际标准，基本形成了一个较为完善的标准化体系，如图 1-16 所示。但从规范软件工程开发过程相关工作的角度出发，尚需进一步完善。

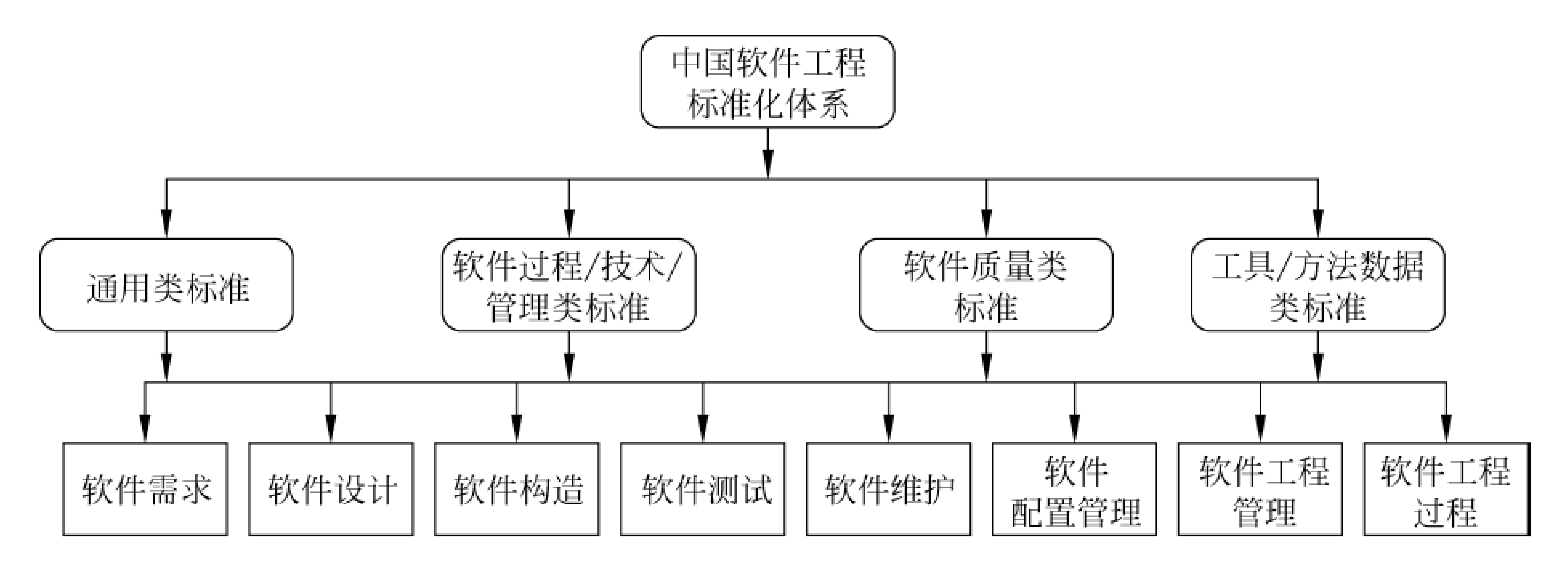


图 1-16 中国软件工程标准化体系

1.5 计算机辅助软件工程

工具在人类生活和生产活动中的地位和作用是众所周知的，对软件工程也不例外。为支持软件开发、维护、管理而研制的计算机程序系统称为软件工具，例如，操作系统、正文编辑器、编译程序系统等。但是，孤立的软件工具只能支持软件工程的某一项活动。各种软件工具的数据结构不统一，程序界面不标准，软件工程引用和加工的数据需要进行格式转换，相关的软件工具因设计标准不一致很难集成为一个软件系统，严重影响了软件开发和维护的效率。

在软件工程活动中，软件工程师和管理员按照软件工程的方法和原则，借助于计算机及其软件工具的帮助，开发、维护、管理软件产品的过程，称为计算机辅助软件工程(Computer-Aided Software Engineering，CASE)。

CASE 是自 20 世纪 60 年代末软件工程的概念(要求采用工程的原则方法和技术开发、维护软件)提出来后，在 20 世纪 80 年代初期出现的被软件工程界普遍接受的术语并作为软件开发自动化支持的代名词。因此，可以简单地把 CASE 理解为：CASE＝软件工程＋自动化工具。狭义地讲，CASE 是一组工具和方法的集合，可以辅助软件生存周期各阶段的开发工作。广义地讲，CASE 是辅助软件开发的任何计算机技术，包括两个含义：一是在软件开发和维护过程中提供计算机辅助支持，二是在软件开发和维护过程中引入工程化方法。

从学术角度来看，CASE 吸收了 CAD(计算机辅助设计)、操作系统、数据库、计算机网络等许多研究领域的原理和技术，把软件开发技术、方法和工具集成为一个统一的整体。而从软件产业的角度来看，CASE 是多种软件开发及系统集成产品与软件工具的集合。其中

的软件工具不是对任何软件开发方法的取代，而是对它们的支持。随着 CASE 术语的出现，可能会出现术语“软件工具”和“CASE 工具”的混淆。严格地说，CASE 工具是除操作系统之外的所有软件工具的总称。软件工具是支持软件开发、维护与移植等的程序系统。

对于大型复杂软件系统的开发来说，若没有 CASE 工具的支持，标准规范难以得到彻底执行，所以软件开发组织要告别“软件作坊”的软件开发模式，进入“软件工厂”的理想开发模式，建立集成化的软件开发环境是十分必要的。而在软件开发过程中，合理利用合适的 CASE 工具能给开发者带来很多好处：提高软件开发效率，缩短开发时间；利于软件开发过程的标准化；利于产生标准化的软件开发文档；利于软件开发过程的管理，使复杂软件开发过程变为可控制；利于系统的维护与扩充等。

目前的 CASE 工具比较多，从解决问题的类型上可以分为以下 3 类。

(1) 辅助设计工具。如分析和设计工具、原型工具、接口设计和开发工具、编程工具、测试工具等。

(2) 辅助计算工具。如风险分析工具、测量分析工具、项目计划工具等。

(3) 辅助管理工具。如项目管理工具、需求跟踪工具、文档管理工具、质量管理工具、软件配置管理工具等。

从发展来看，CASE 已经从一些具体的开发工具发展成为一种独特的、以自动化环境支持为基础的系统开发方法。就开发方法学的角度来看，采用 CASE 方法时，必须结合一种具体的开发方法，如结构化方法、原型法或面向对象的方法，CASE 方法为这些方法提供专门的支持工具。CASE 方法不是完全独立的，它可用于支持结构化方法，也可用于支持原型方法和面向对象方法。

1.6 软件工程人员的职业道德与行为准则

和其他工程人员一样，软件工程人员必须承认他们的工作不仅仅是技术的应用，还要担负许多责任。他们的工作是在法律和社会的框架内完成的：软件工程要受地方的、国家的、国际的各种法律的约束，因而工程人员要想受人尊敬，其行为就必须合乎道德，必须有责任心。

软件工程人员必须坚持诚实正直的行为准则，这是不言而喻的；他们不能用掌握的知识和技能做不诚实的事情，更不能给软件工程行业抹黑。然而，在有些方面，某些行为没有法律加以规范，只能靠职业道德来约束，虽然这种约束有时是软弱无力的，包括以下几个方面。

(1) 机密。工程人员必须严格保守雇主或客户的机密，而不管是否签署保密协议。

(2) 工作能力。工程人员应该实事求是地表述自己的工作能力，不应有意接受超出自己能力的工作。

(3) 知识产权。工程人员应当知晓控制专利权、著作权等知识产权使用的地方法律，必须谨慎行事，确保雇主和客户的知识产权受到保护。

(4) 计算机滥用。软件工程人员不应运用自己的技能滥用他人的计算机。滥用计算机有时对他人影响不大，但有些时候后果非常严重(如泄露个人隐私和传播病毒)。

ACM/IEEE 等组织还颁布了职业行为准则或职业道德准则，凡是加入这些组织的成员

必须严格遵守，这些行为准则只涉及基本的道德行为。内容如下[①]：

(1) 公众感。软件工程人员应始终与公众利益保持一致。

(2) 客户与雇主。软件工程人员应当在与公众利益保持一致的前提下，满足客户与雇主的最大利益。

(3) 产品。软件工程人员应当保证他们的产品及其相关附件达到尽可能高的行业标准。

(4) 判断力。软件工程人员应当具有公正和独立的职业判断力。

(5) 管理。软件工程管理者和领导者应当拥护并倡导合乎道德的有关软件开发和维护的管理方法。

(6) 职业感。软件工程人员应当弘扬职业正义感和荣誉感，尊重社会公正利益。

(7) 同事。软件工程人员应当公平地对待和协助每一位同事。

(8) 自己。软件工程人员应当毕生学习专业知识，倡导合乎职业道德的职业活动方式。

准则的简写版把对软件工程人员的要求做了高度抽象性的概括。较长版本中的条款把这些要求细化，并给出了实例，用于规范软件工程专业人员的工作方式。如果没有这些总体要求，所有的细节都是教条而枯燥的；而没有这些细节，总体要求就会变成空洞的高调。

本行为准则包括 8 项基本原则，针对包括软件工程行业的从业者、教育者、管理者、监督者、政策制定者、接受培训者和学生在内的职业软件工程人员。这 8 条原则阐明了个人、团队和机构之间职业道德上的责任关系以及他们在其中应该履行的基本义务。每一原则的条款都表述了这些关系中的一些义务。这些义务既基于软件工程人员的人性，也对那些受他们的工作和软件工程实践的独特环境影响的人们表示出特别的关怀。本准则把这些内容规定在任何一个自称为或渴望成为软件工程人员的义务中。

本 章 小 结

软件工程是一门指导软件开发的工程学科，它以计算机理论及其他相关学科的理论为指导，采用工程化的概念、原理、技术和方法进行软件的开发和维护。软件工程学的主要内容是软件开发技术和软件工程管理两个方面。本章主要对软件工程学的整体内容进行了简要的描述，以使学习者对本门学科有一个概要性的了解。

软件工程研究的主要内容有方法、工具和过程 3 个要素，它们构成了一种层次化的技术方法提供如何构造软件的技术。工具为方法和语言提供自动化或半自动化的支持。软件工程的过程是黏结剂，把方法和工具黏结在一起。整个体系结构反映了以质量为中心的观点。关注质量是其根本出发点和最终目标。

从软件危机被提出以来，人们一直在寻找解决它的方法。软件方法学是从各种不同角度、不同思路去认识软件的本质，如结构化的程序设计，面向对象的开发，CMM，UML 等。现在，软件工程较以往已有了长足的进步，如软件开发过程、开发方法、质量保证、项目管理及工具环境等方面都变得成熟起来。但是，要成为一个完全成熟的学科还需要做大量的工

① Sommerville. 软件工程. 程成，等译. 北京：机械工业出版社，2003

作。复杂性是软件的本质原因，现在还没有任何一种方法（或称“银弹”）能够用来解决软件危机中的所有问题。

软件工程的基础建立在多个不同的学科之上。软件工程学科的理论基础是数学和计算机科学。它又是一门交叉性的工程学科，涉及工程科学、管理学等。本章还介绍了 ACM 与 IEEE Computer Society 提出的软件工程知识体系 SWEBOK，对于整体把握软件工程学科提供了可参照的方案。软件专业的学生必须掌握该学科的基本技能和知识，具备理论与实践相结合的能力，并充分意识到优秀设计的价值。同时还需要了解与学习计算学科以外的领域知识，做到能够支持该领域的软件开发。

思考与练习

1. 什么是软件？它有哪些特点？软件的本质是什么？

2. 软件的复杂性表现在哪些方面，有没有解决软件危机的“银弹”存在？

3. 随着软件的广泛应用，公众所面临的可能危险（由于软件的错误引起）受到越来越多的关注。试举一个因为计算机程序的失败而带来巨大灾难的（对人类或经济均可）实际可能发生的场景。

4. 什么是工程？为何将软件的开发定义为一门工程？深刻理解 IEEE 给出的软件工程定义。

5. 软件工程研究的主要内容有方法、工具和过程 3 个要素，它们之间的关系如何？

6. 简述面向过程的结构化开发方法与面向对象的开发方法。

7. 软件工程是一门交叉性的工程学科，简述软件工程与其他学科间的关系。

8. IEEE/ACM 对计算学科的定义与学科的划分有何重要意义？

9. 软件工程的知识体系（SWEBOK）包括哪些知识域？

10. 在过去的若干年中，发布了很多软件工程的标准和规范，请说出几种常见的软件工程的国际标准与体系。

11. 和其他工程人员一样，软件工程人员的工作是在法律和社会的框架内完成的，还要遵守相应的职业道德来约束自己的行为。你认为软件工程人员的职业道德应包括哪些内容？

第2章 软件过程

软件过程是指软件的整个生命周期，从需求获取、需求分析、设计、实现到发布的一个过程模型。一个软件过程定义了软件开发中采用的方法，但软件工程还包含该过程中应用的技术——技术方法和自动化工具。

——维基百科(Wikipedia)

软件过程(Software Process)是指人们用于开发和维护软件及其相关产品的一系列活动、方法和实践，包括软件工程活动和软件管理活动。有效的软件过程能够将人员、工具和方法进行有机的结合。任何一个软件组织都需要一个完善的软件过程管理和持续改进的机制，从而保证组织的软件过程能力不断地得到提高。

2.1 软件过程

2.1.1 过程及其特征

提供一项服务或建造一个产品，无论是准备一次会议，讲授一门课程，或者进行一次旅行，我们总是按照一系列步骤来完成一套任务，这些任务每次总是按同样的次序来执行。

IEEE(STD-610)将过程(Process)定义为实现给定目标所执行的一系列操作步骤。由此，我们可以把一个有序任务集合看做是一个过程，一个用来产生某类想要的产品所涉及的活动、约束和资源的步骤序列。

一般而言，过程具有如下的一些特征(图 2-1)。

每个过程均包含一系列的阶段。例如，统一软件开发过程(RUP)中的软件生命周期在时间上被分解为初始阶段、细化阶段、构造阶段和交付阶段 4 个阶段。每个阶段结束于一个主要的里程碑(Milestones)，每个阶段本质上是两个里程碑之间的时间跨度。在每个阶段的结尾执行一次评估以确定这个阶段的目标是否已经满足。如果评估结果令人满意，可以允许项目进入下一个阶段。



图 2-1 软件过程的概念

过程是一个生命周期。当过程涉及某种产品的建立时，称这个过程为一个生命周期(Life Cycle)。因此，软件开发过程又可以称为软件生命周期(Software Life Cycle)，因为它描述了一个软件产品的生命：从它的需求、建模开始到软

件的构建和发布。

每个过程均有一系列已定义好的输入作为其操作的对应。过程的目的在于生产出产品,为此,它必须有一系列的输入。每个过程均会形成一系列的输出,并以此作为其他过程的输入,过程会将相应的输入转换为事先已定义好的输出。软件生存期各个阶段的问题不是孤立的,而是相互影响、相互依存的。每一阶段的工作成果将成为下一阶段工作的基础,后一阶段发现的问题也应追溯到前一阶段去找原因,这种前后相承的关系也会带来错误的传递。

过程具有迭代特征。一次迭代是一个完整的开发循环,在过程中的每一次顺序的通过称为一次迭代,因此一个过程迭代是在过程中所有阶段(活动)的一次完整的经过(图 2-2)。每个阶段可以进一步分解为更细的迭代,通过不断细化来加深对问题的理解和对产品的增量开发。软件生命周期是迭代的连续,称为一个迭代生命周期。迭代不是简单的重复,而是包括了生成一个可执行版本的开发活动,以实现软件的递增式的开发。

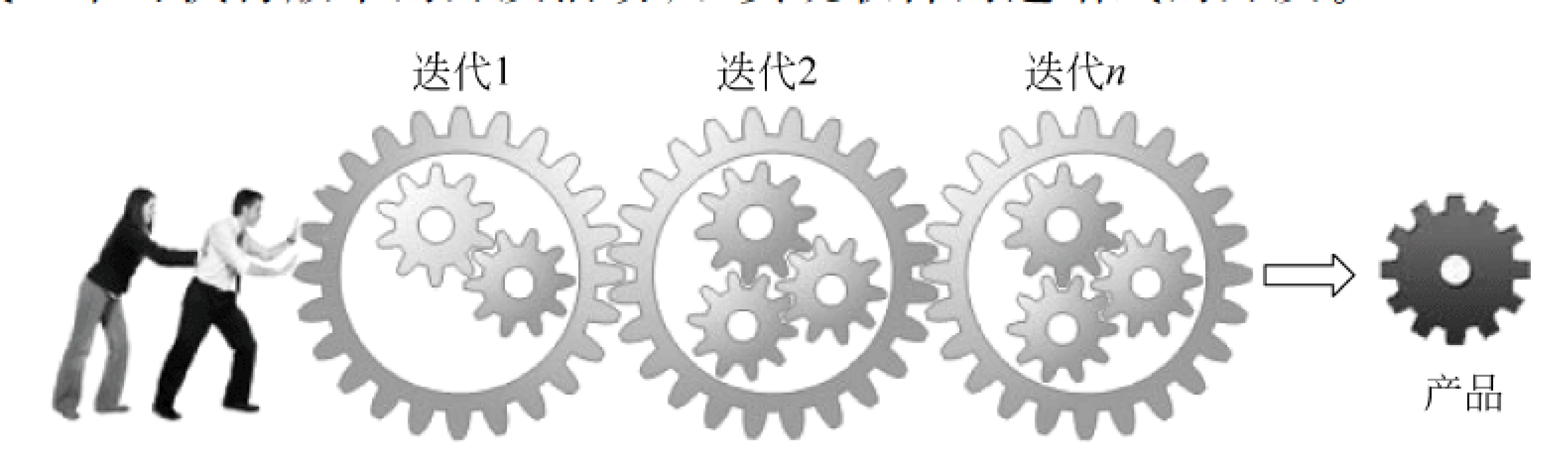


图 2-2 过程迭代的基本概念

在软件工程的三要素中,软件过程将人员、方法、工具和管理有机结合,形成一个能有效控制软件开发质量的运行机制。

2.1.2 软件过程的公共框架

根据 ISO/IEC 12207 标准的定义,软件过程是指软件生命期中的若干活动的集合。活动,又称为工作流程(Workflow),可细分为任务。使用软件过程的目的是为了在一定的时间和经费预算内开发出高质量的产品。一个好的软件过程可以提高软件开发组织的生产效率、提高软件质量、降低成本并减少风险。

软件过程提供了一个公共框架,在该框架下可以建立一个软件开发的综合计划(图 2-3)。

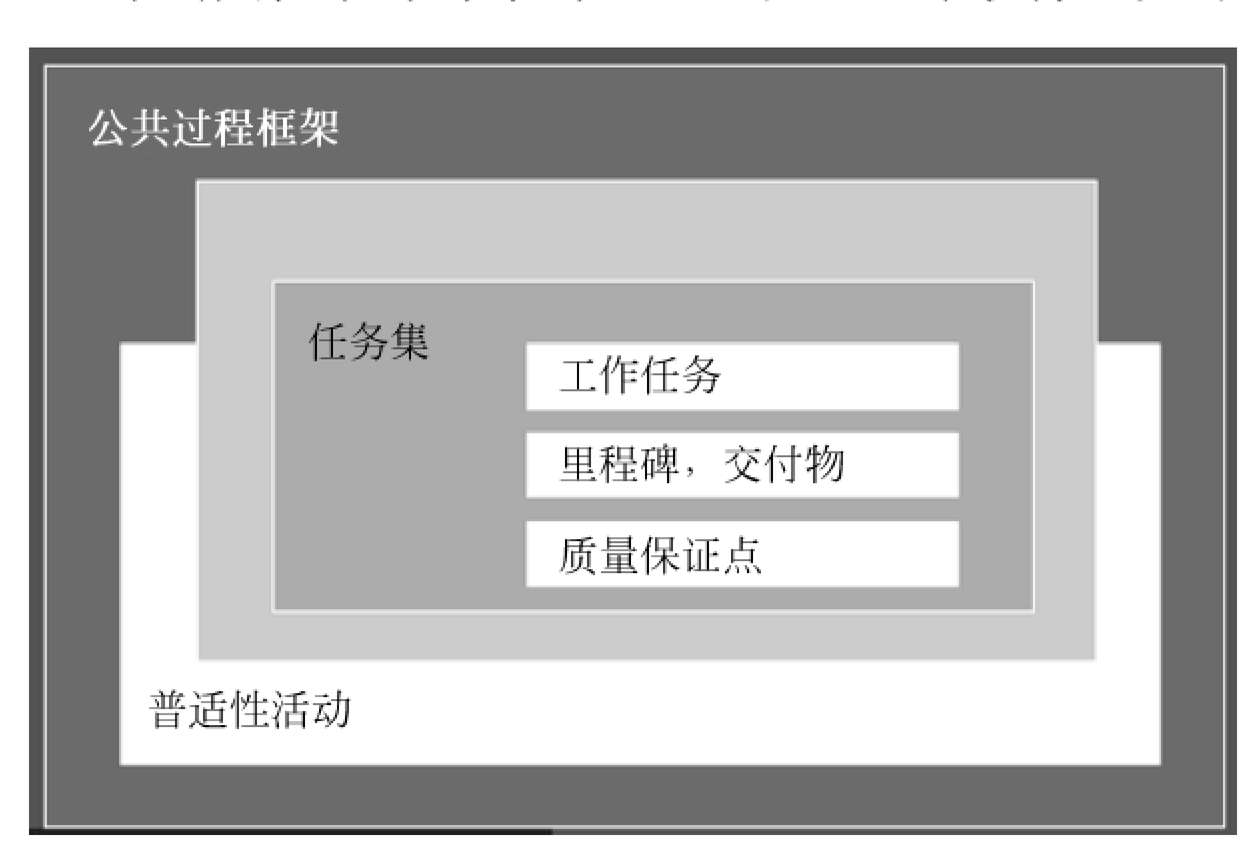


图 2-3 软件过程的公共框架

(1) 若干框架活动适用于所有软件项目,而不在乎其规模和复杂性。

(2) 若干不同任务的集合——每一个集合都由工作任务、里程碑、交付物以及质量保证点组成——使框架活动适应不同软件项目的特征和项目组的需求。

(3) 若干保护性活动——如软件质量保证、软件配置管理、测试与度量等。保护性活动独立于任何一个框架活动,且贯穿于整个过程之中。

2.2 软件过程模型

2.2.1 理解软件过程模型

为解决实际应用中的问题,人们总结出了很多软件开发策略和方法,称为软件过程模型(Software Process Model)。

软件过程模型是软件开发的指导思想和全局性框架,它的提出和发展反映了人们对软件过程的某种认识观,体现了人们对软件过程认识的提高和飞跃。软件过程模型是从一个特定角度提出的软件过程的简化描述,是一种开发策略,这种策略针对软件工程的各个阶段提供了一套范型,使工程的进展达到预期的目的。其中每个过程模型都代表了一种将本质上无序的活动转换为有序化的步骤,每个模型都具有能够指导实际软件项目的控制及协调的特性。

随着需求不断增加,软件项目规模也越来越大,人们逐渐意识到要建立一种抽象模型来对软件开发过程进行描述。模型的本质就在于简单化。软件过程模型就是对被描述的实际过程的抽象,它包括构成软件过程的各种活动、软件产品以及软件工程参与人员的不同角色。对一个软件的开发无论其大小,都需要选择一个合适的软件过程模型,这种选择基于项目和应用的性质、采用的方法、需要的控制,以及要交付的产品的特点。一个错误模型的选择,将导致开发方向迷失。

自 20 世纪 60 年代以来,软件工程思想逐渐形成与发展,出现了很多软件过程模型与方法,例如瀑布模型、增量模型和螺旋模型等,我们称它们为传统软件过程模型。这些模型的出现很好地解决了当时软件开发过程的各类问题,使软件开发的小作坊式的随意开发变得日益规范起来。人们在不断地改进传统软件过程模型的同时,新的模型和方法也不断地涌现。以“敏捷过程”、“极限编程”、“净室软件工程”等为代表的新的过程模型被越来越多地运用到日常的开发工作中去。这些模型的提出不断丰富着软件过程理论,也为开发者提供了一个可参考的过程框架。但是,这些新方法或多或少仍有其局限性,这也是激励我们对现有的软件过程及其模型进行持续地改进的原动力。

2.2.2 瀑布模型

瀑布模型(The Waterfall Model)也称为线性顺序模型,是由 Royce 在 1970 年最初提出的软件开发模型。在瀑布模型中(图 2-4),开发被认为是按照需求分析、设计、实现(编码)、测试和维护阶段顺序地进行,当线性序列完成之后就能够交付一个完善的系统。直到 20 世纪 80 年代初,瀑布模型是唯一被广泛接受的生命周期模型。

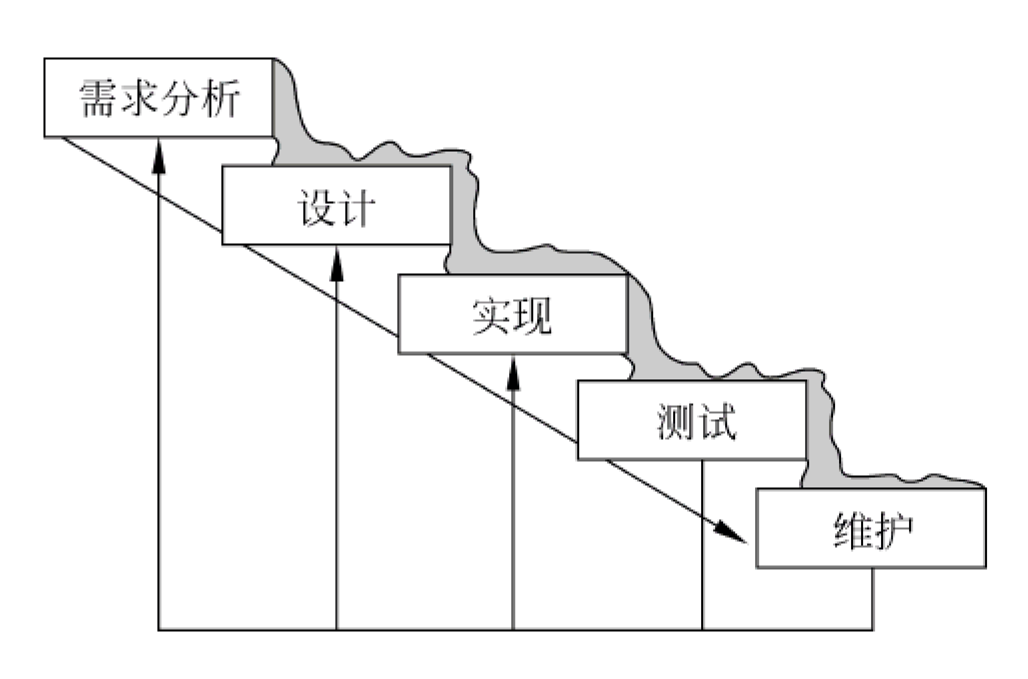


图 2-4 瀑布模型

瀑布模型强调系统开发应有完整的周期，且必须完整地经历周期中的每一个开发阶段。由于该模式强调系统开发过程需有完整的规划、分析、设计、测试及文件等管理与控制，因此能有效地确保系统品质，它已经成为业界大多数软件开发的标准。

"线性"是人们最容易掌握并能熟练应用的思维方法。当人们碰到一个复杂的"非线性"问题时，总是千方百计地将其分解或转化为一系列简单的线性问题，然后逐个解决。然而在实践中，过程很少能够以纯线性的方式进行。应提倡以一种迭代的方式重复地使用瀑布模型，通过回到前面的阶段或改变前一阶段的结果的迭代是非常普遍的。但是，很多人忽视了这一点。

线性顺序模型过程的缺点也是非常明显的，主要有以下几点。

(1) 实际的项目很少按照该模型给出的顺序进行。

(2) 项目初期用户常常难以清楚地给出所有需求，而这恰恰是线性顺序模型所必须给出的。

(3) 用户必须有耐心，程序的运行版本要等到项目开发晚期才能得到。大的错误如果到检查运行程序时才被发现，后果可能是灾难性的。

(4) 开发者常常被不必要地耽搁。项目组某些成员不得不等待组内其他成员先完成其依赖的任务。

尽管如此，瀑布模型仍然是软件工程中应用最广泛的过程模型。很显然，它比起软件开发中随意的状态要好得多。

2.2.3 演化软件过程模型

人们已经越来越认识到软件就像所有复杂系统一样要经过一段时间的演化。业务和产品需求随着开发的发展常常发生改变，想找到最终产品的一条直线路径是不可能的。

演化模型是利用一种迭代的思想方法，它的特征是使软件工程师渐进地开发逐步完善的软件版本。主要包括增量模型和螺旋模型两种范型。

1. 增量模型

如同房子的建造一样，软件也是一步一步构建出来的。当一个软件产品正处在开发过程中时，每一步都在它前一步的基础上进行。例如，在某一时段扩充了设计，另一个时段对

模块编写代码，以这种渐渐增加的方式构造软件产品，直到完成(图 2-5)。当然，并不是每天都有进展。就像建造房屋偶尔需要拆掉位置不对的一段墙或一扇窗户一样，有时有必要重新确定规格说明，重新设计，重新编写代码，甚至抛弃已完成的全部工作重新开始。但是产品有时间歇地发展的情况并不能否定软件产品是逐渐地构建造的这个基本事实。



图 2-5 增量的概念

软件在工程上渐增实现的事实，导致一个利用这个软件开发特性的模型的发展，即如图 2-6 所示的“增量模型”(The Incremental Model)。产品是以一系列增量构件的形式设计、实现、集成和测试的，其中，每个构件由一些代码块组成，这些代码块来自多个相互作用的模块，完成特定的功能。增量模型是一种非整体开发的模型。根据增量的方式和形式的不同，分为基于瀑布模型的渐增模型和基于原型的快速原型模型。一般的增量模型如图 2-6 所示。该模型具有较大的灵活性，适合软件需求不明确，设计方案有一定风险的软件项目。

增量模型和瀑布模型之间的本质区别是：瀑布模型属于整体开发模型，它规定在开始下一个阶段的工作之前，必须完成前一个阶段的所有细节；而增量模型属于非整体开发模型，它推迟某些阶段或所有阶段小的细节，从而较早地产生工作软件。

增量模型融合了线性顺序模型的基本成分并具有迭代特征(重复地应用线性模型和原型)。增量模型采用随着时间的进展而交错的线性序列，每个线性序列产生的软件版本称为发布的“增量”。增量模型强调每个增量均发布一个可操作产品，它确实给用户提供了一定的功能，并且提供给用户评估最终交付软件的一个平台。

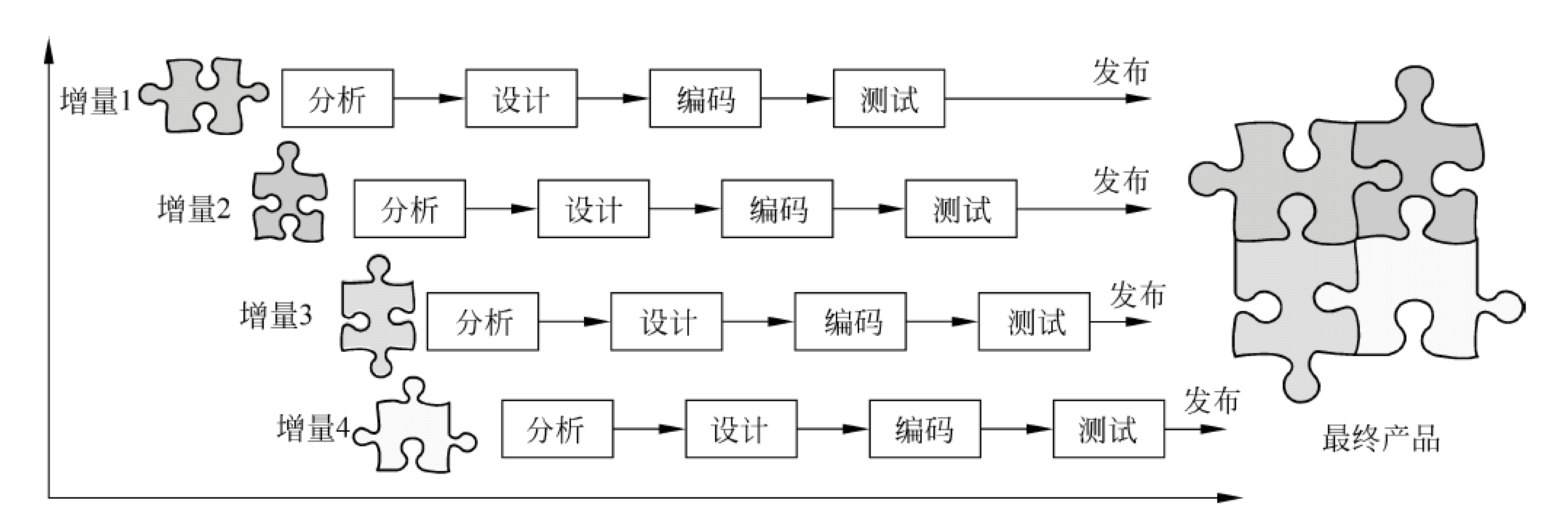


图 2-6 增量模型

在增量模型开发过程中，客户大致地描述系统需提供的功能，并指明哪些功能是重要的，哪些是相对不重要的。开发人员据此确定开发过程中的系列交付增量，每个增量提供系统功能的一个子集。对增量中功能的分配取决于用户指明的功能优先次序。一旦确定了系统增量，第一个增量将要详细地定义功能的需求，并用最合适的开发过程来

开发。在开发的同时,为稍后的增量准备的需求分析开始进行,但不对目前增量的需求做出变更。

增量模型的另一个优点是,逐步增加产品功能可以使用户有较充裕的时间学习和适应新产品,从而减少一个全新的软件可能给客户组织带来的冲击。使用增量模型的困难是,在把每个新的增量构件集成到现有软件体系结构中时,必须不破坏原来已经开发出的产品。此外,必须把软件的体系结构设计得便于按这种方式进行扩充,向现有产品中加入新构件的过程必须简单、方便,也就是说,软件体系结构必须是开放的。但是,从长远观点看,具有开放结构的软件拥有真正的优势,这样的软件的可维护性明显好于封闭结构的软件。因此,尽管采用增量模型比采用瀑布模型和快速原型模型需要更精心的设计,但在设计阶段多付出的劳动将在维护阶段获得回报。

2. 螺旋模型

软件开发过程几乎都存在风险,例如,开发团队的核心人员突然离职;软件开发者可能因为预订的硬件未到位造成承诺的软件不能交付,以致最终破产;在投入了大量资金开发一个主要的软件产品之后,在没有新的资金投入而使开发工作处于停顿;当产品即将完成时,另一个新技术上的突破可能使整个产品价值全无;当辛苦数年研发了一个数据库管理系统,但在产品面市前,竞争对手可能抢先上市了价格更便宜且功能相当的软件。鉴于上述这些因素,软件开发者应该尽力将这种风险减到最小。

对于大型软件,只开发一个原型往往达不到要求。螺旋模型将瀑布模型和增量模型结合起来,并加入了风险分析。该模型将开发过程划分为沟通、制定计划、风险分析、实施工程、构造与发布和系统评估 6 个活动,如图 2-7 所示。

沿着螺旋线每转一圈,表示开发出一个更完善的新的软件版本。如果开发风险过大,开发机构和客户无法接受,项目有可能就此中止;多数情况下,开发过程会沿着螺旋线继续下去,自内向外逐步延伸最终得到满意的软件产品。

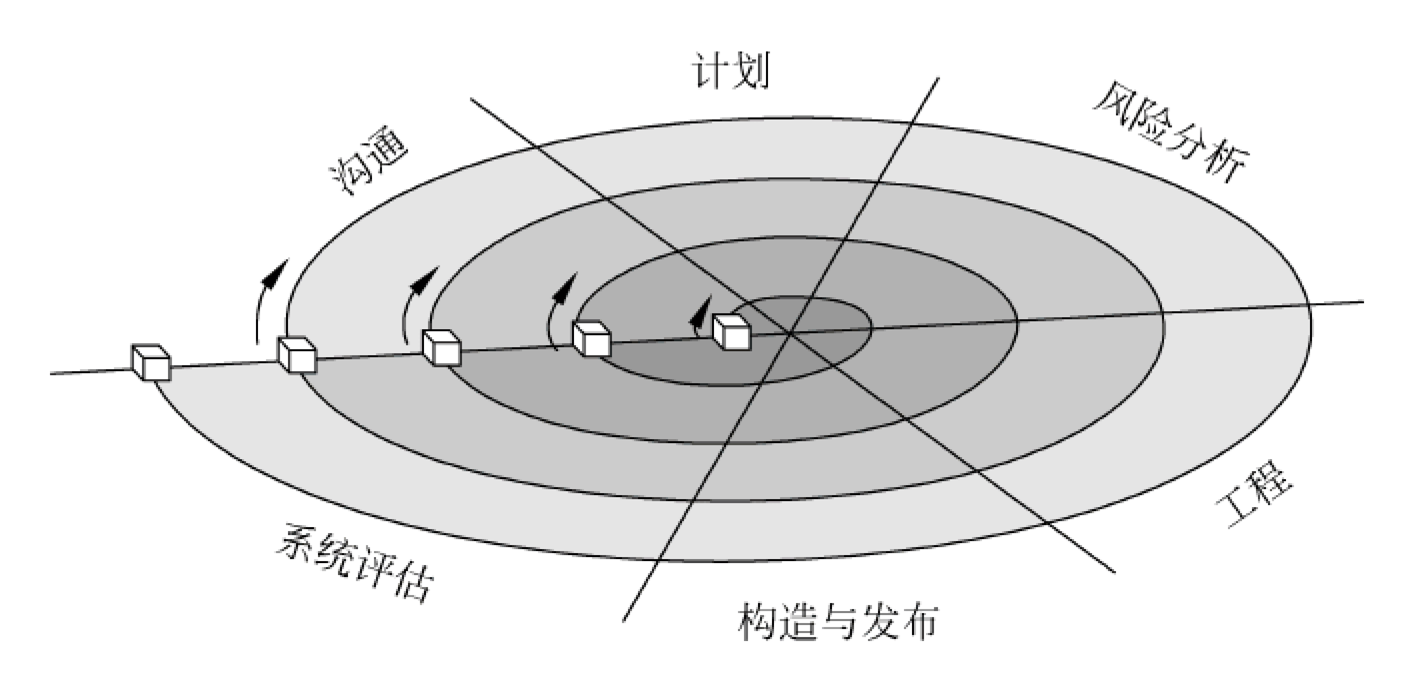


图 2-7 螺旋模型中的螺旋周期

今天,微软公司典型项目的生命周期包括 3 个阶段:计划阶段完成功能的说明和进度表的最后制定;开发阶段写出完整的源代码;稳定化阶段完成产品,使之能够批量生产。微软公司把这 3 个阶段称为“进度表完成及项目计划被批准”、“代码完成”以及“发布供生产”(软件产品上的“生产”概念是指磁盘和文档的复制)。这 3 个大阶段以及阶段间内在的

循环方法与循序的"瀑布"式生命周期很不相同，后者是由需求、详细设计、模块化的代码设计与测试、集成测试以及系统测试组成。微软公司的3个阶段更像风险驱动的、渐进的"螺旋"生命周期模型，该模型已日益为软件开发机构所采用[①]。

2.2.4 快速原型开发方法

原型开发的思想来源于工程实践。快速原型(Rapid Prototype Model)是利用原型辅助产品设计开发的一种新思想。经过简单快速分析，快速实现一个原型，用户与开发者在试用原型过程中加强通信与反馈，通过反复评价和改进原型，减少误解，弥补漏洞，适应变化，最终提高软件质量。快速原型方法在建筑设计、CAD/CAM、工业产品设计等领域都得到了广泛的应用。

在软件的开发过程中，由于种种原因，在需求分析阶段得到完全、一致、准确、合理的需求说明是很困难的。在获得一组基本需求说明后，就应快速地使其"实现"，通过原型反馈，加深对系统的理解，并满足用户基本要求，使用户在试用过程中受到启发，对需求说明进行补充和精确化，消除不协调的系统需求，逐步确定各种需求，从而获得合理、协调一致、无歧义的、完整的、现实可行的需求说明。目前，快速原型思想已逐步应用到软件开发的其他阶段，向软件开发的全过程扩展。即先用相对少的成本，较短的周期开发一个简单的，但可以运行的系统原型给用户演示或让用户试用，以便及早澄清并检验一些主要设计策略，在此基础上再开发实际的软件系统。

快速原型方法的开发步骤如图2-8所示。



图2-8 快速原型开发步骤

(1) 快速分析。在分析人员与用户密切配合下，迅速确定系统的基本需求，根据原型所要体现的特征描述基本需求以满足开发原型的需要。

(2) 构造原型。在快速分析的基础上，根据基本需求说明尽快实现一个可行的系统。这里要求具有强有力的软件工具的支持，主要考虑原型系统能够充分反映所要评价的特性，并忽略最终系统在某些细节上的要求。

(3) 运行与评价。在运行的基础上，考核评价原型的特性，分析运行效果是否满足用户的愿望，纠正过去交互中的误解与分析中的错误，增添新的要求，并满足因环境变化或用户的新想法引起的系统要求变动，提出全面的修改意见。

(4) 修改原型。根据评价原型的活动结果进行修改。若原型未满足需求说明的要求，说明对需求说明存在不一致的理解或实现方案不够合理，则需要根据明确的要求迅速修改原型。

(5) 最终系统实现。快速原型的一个基本特性是体现"快"字。开发者应该尽可能快地建造原型，以加快软件开发进程。因此，快速原型的内部结构无关紧要，最重要的是快速建造原型并快速修改以反映客户的需求。所以，速度是关键。

原型方法不应只被看做是技术工具，它是重要的管理和沟通工具。用原型提供关于某

---

① Michael A. Cusumano. 微软的秘密. 北京：北京大学出版社，1996：121

类风险的信息非常有效。例如，时间限制通常可通过构建一个模型并计算原型是否能达到必需的性能来进行测试。如果原型是产品相关特性的准确的功能体现，则在原型上进行的测量应当能够清楚地告诉开发者时间要求能否达到。在微软公司的软件开发过程中，当程序经理具体说明一件新产品或一个新版本时，构造原型便成为他们的基本行为。这从许多方面来说都使开发前测试成为可能，尤其是在可用性方面，有助于对与用户交互情景做出更好的理解，它也能使产品说明更紧凑[①]。

目前广泛地使用第 4 代语言（4GT）构建快速原型，当快速原型的某个部分是利用软件工具由计算机自动生成时，可以把这部分用到最终的软件产品中。例如，用户界面通常是快速原型的一个关键部分，当使用屏幕生成程序和报表生成程序自动生成用户界面时，实际上可以把得到的用户界面用在最终的软件产品中。

2.2.5 统一软件过程

统一软件过程（Rational Unified Process，RUP）是一个二维的软件开发模型，如图 2-9 所示。横轴各阶段以时间坐标组织，具有过程展开的生命周期特征，体现开发过程的动态结构，用来描述它的术语主要包括周期（Cycle）、阶段（Phase）、迭代（Iteration）和里程碑（Milestone）；纵轴以内容来组织为自然的逻辑活动，体现开发过程的静态结构，用来描述它的术语主要包括活动（Activity）、产物（Artifact）、工作者（Worker）和工作流（Workflow）。

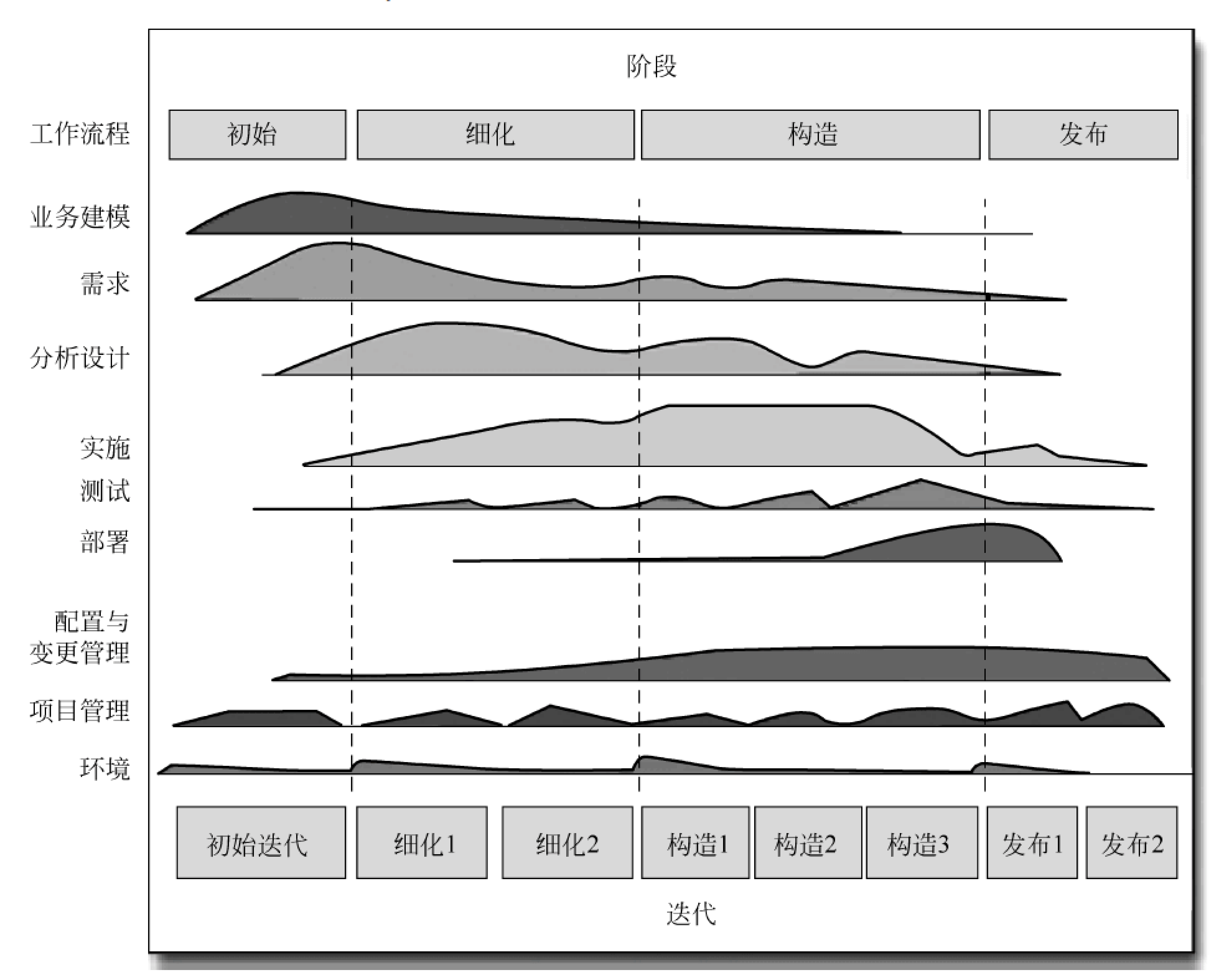


图 2-9 统一软件过程

① Michael A. Cusumano. 微软的秘密. 北京：北京大学出版社，1996：137

1. RUP 的阶段和里程碑

在 RUP 模型中,时间维从组织管理的角度描述整个软件开发的生命周期,是 RUP 的动态组成部分。RUP 中的软件生命周期在时间上被分解为 4 个顺序的阶段,分别是初始阶段、细化阶段、构造阶段和交付阶段,如图 2-10 所示。每个阶段结束于一个主要的里程碑。

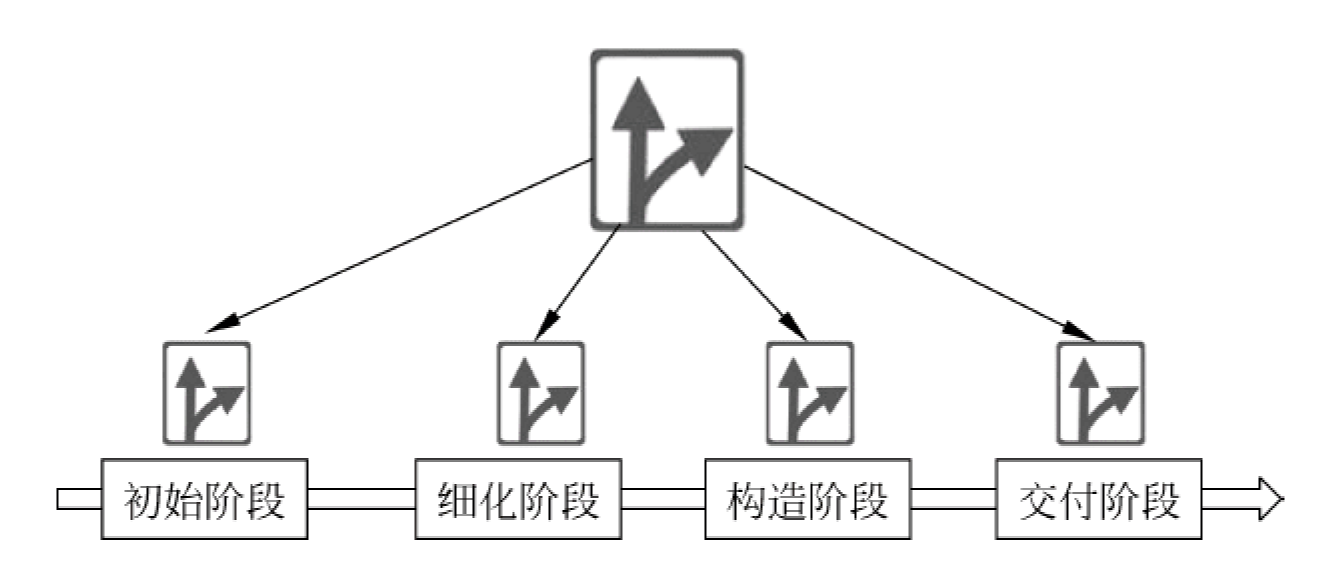


图 2-10 RUP 的阶段和主要里程碑

初始阶段的目标是为系统建立业务用例和确定项目的边界。为了达到该目的必须识别所有与系统交互的外部实体,在较高层次上定义交互的特性。它包括识别所有用例和描述一些重要的用例。业务用例包括验收规范、风险评估、所需资源估计、体现主要里程碑日期的阶段计划。

细化阶段的目标是分析问题领域,建立健全的体系结构基础,编制项目计划并淘汰项目中最高风险的元素。本阶段的主要目标是确保软件结构、需求、计划足够稳定;确保项目风险已经降低到能够预计完成整个项目的成本和日程的程度;针对项目的软件结构上的主要风险已经解决或处理完成;通过完成软件结构上的主要场景建立软件体系结构的基线;建立一个包含高质量组件的可演化的产品原型。

在构造阶段,所有剩余的构件和应用程序功能被开发并集成为产品,所有的功能被详细地测试。在构造阶段,从某种意义上说,是重点在管理资源和控制运作以优化成本、日程、质量的生产过程。

交付阶段的目的是将软件产品交付给用户群体。只要产品发布给最终用户问题常常就会出现:要求开发新版本,纠正问题或完成被推迟的问题。该阶段包括若干重复过程,包括 Beta 版本、通用版本、Bug 修补版和增强版。相当大的工作量消耗在开发面向用户的文档,培训用户,在初始产品使用时,支持用户并处理用户的反馈上。

2. RUP 的迭代过程

在 RUP 中,迭代被定义为:包括产生产品发布(稳定、可执行的产品版本)的全部开发活动和要使用该发布必需的所有其他外围元素。所以,在某种程度上,开发迭代是一次完整地经过所有工作流程的过程:(至少包括)需求工作流程、分析设计工作流程、实施工作流程和测试工作流程。实质上,它类似小型的瀑布式项目。RUP 认为,所有的阶段(需求及其他)都可以细分为迭代。每一次的迭代都会产生一个可以发布的产品,这个产品是最终产品的一个子集。迭代的思想如图 2-11 所示。

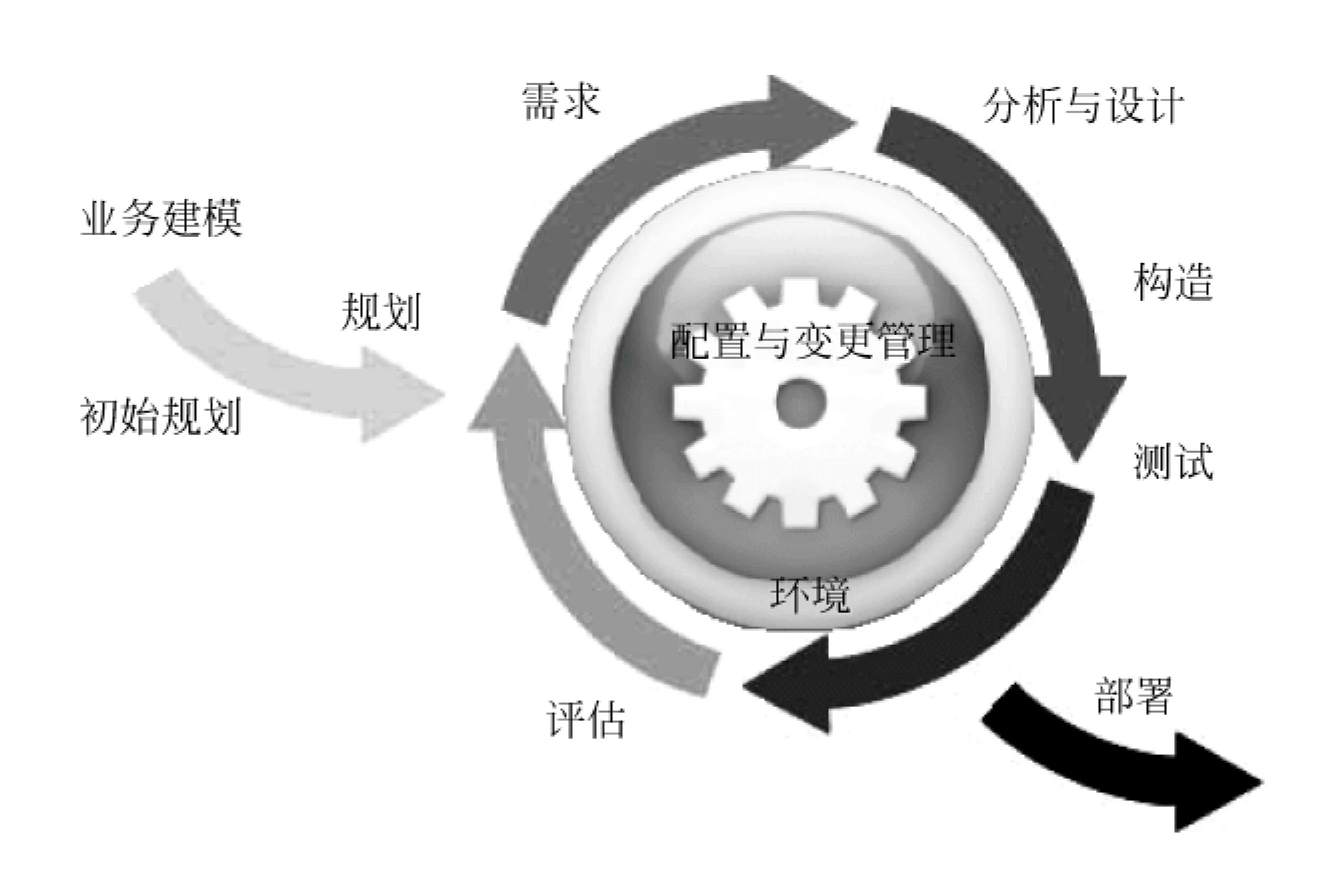


图 2-11 RUP 的迭代过程

2.2.6 核心工作流

工作流是产生具有观察力结果的活动系列。图 2-9 中描述了 9 个不同的工作流，这些工作流在整个开发周期中一次又一次被访问，在每一次迭代中以不同的重点和强度重复。

(1) 建模工作流程的任务是为系统建立业务模型，业务模型是该工作流程最主要的产品，业务建模的目的是通过对业务模型的建立，理解需要开发软件系统的组织的结构和动态行为，确保用户、最终使用者和开发人员对组织有共同的理解，为下一步获取系统需求打基础。

(2) 需求工作流的任务是描述系统的需求，定义系统的开发范围，最主要的产品是用例(Use Case)模型。

(3) 分析和设计工作流的任务是将需求转换为描述怎样进行系统实现的规格说明，显示系统是"如何"在实现阶段被"实现"的。

(4) 实现工作流的目的包括以层次化的子系统形式定义代码的组织结构；以组件的形式(源文件、二进制文件、可执行文件)实现类和对象；将开发出的组件作为单元进行测试以及集成由单个开发者(或小组)所产生的结果，使其成为可执行的系统。

(5) 测试工作流程进行系统测试和集成测试，检验需求是否全部被实现，在交付软件前标识全部缺陷。

(6) 部署工作流的目的是成功地生成版本并将软件分发给最终用户。部署工作流描述了那些与确保软件产品对最终用户具有可用性相关的活动，包括软件打包、生成软件本身以外的产品、安装软件、为用户提供帮助。

(7) 项目管理工作流并没有意图覆盖项目管理的所有方面，主要有以下 3 个目的：为管理软件密集型项目提供框架；为计划、执行、监督项目和分配管理人员提供实际的指南；为管理风险提供框架。

(8) 配置和变更管理工作流描绘了如何在多个成员组成的项目中控制大量的产物。配置和变更管理工作流提供了准则来管理演化系统中的多个变体，跟踪软件创建过程中的版本。工作流描述了如何管理并行开发、分布式开发，以及如何自动化创建工程，同时也阐述

了对产品的修改原因、时间、人员保持审计记录等。

(9) 环境工作流程的任务是为开发组织配置开发环境,可能涉及的活动有过程配置、过程实现、选择工具、自行开发工具、技术支持和培训。

2.2.7 形式化方法模型

在形式化方法模型中,软件需求描述被精炼成用数学符号表达的详细的形式化描述。不同于其他过程模型,设计、实现和单元测试等开发过程被数个形式化转换的开发过程所代替。形式化描述经过系列转换变成执行程序。在转换过程中,系统形式化的数学表达被系统地转换成更详细但数学上仍然是正确的系统表示。每个步骤增加细节直到形式化描述被转换成对等的程序。转换是非常准确的,有严格的数学方法保障,因此,转换的正确性能得到保证。图 2-12 为形式化方法的过程模型。

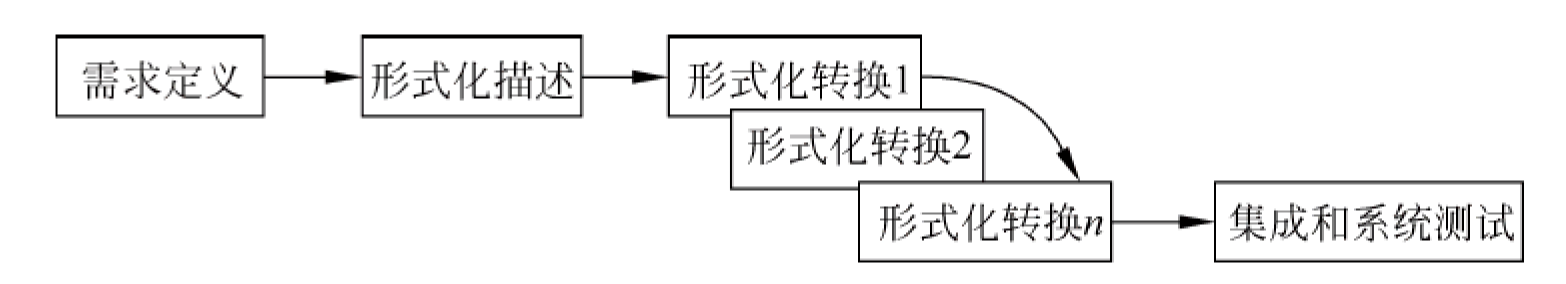


图 2-12 形式化方法的过程模型

由于数学方法具有严密性和准确性,形式化方法开发过程所交付的软件系统具有较少的缺陷和较高的安全性。形式化方法模型最好的例子是净室软件工程,它依赖增量式软件开发,同时在每个阶段都采用形式化方法去开发和验证其正确性。整个开发过程中不存在缺陷测试,系统测试的重心主要集中在评估系统的可靠性上。

但是,形式化方法并不是主流开发方法,在实际软件开发中应用较少,其主要原因是形式化方法的开发还很费时且昂贵,另外难以使用该模型与用户进行沟通。其他原因是现实应用的系统大多数是交互性强的软件,但是这些系统难以用形式化方法进行描述。

2.2.8 软件复用——基于构件的开发方法

复用就是指利用现成的东西,文人称之为"拿来主义"(该词是鲁迅先生首倡的)。被复用的对象可以是有形的物体,也可以是无形的成果。复用不是人类懒惰的表现而是智慧的表现,因为人类总是后人继承了前人的成果,并不断加以利用、改进或创新后才取得进步。

软件复用对于提高开发效率和软件质量有着巨大的发展潜力。软件复用使人们在软件开发中不必"重新发明轮子"或"一切从零开始",在新系统的开发中着重于解决出现的新问题。软件构件技术是支持软件复用的核心技术,是近几年来迅速发展并受到高度重视的一个学科分支。软件复用可以通过恰当的使用构件得以实现,构件化的软件开发主要通过提高软件的重用性来提高软件开发效率。在构件库支持下,软件工程师应该能够"取众家之长",避免重复开发已有的软件。

基于构件的开发(Component-Based Development,CBD)是一种软件开发新范型,它是在一定构件模型的支持下,复用构件库中的一个或多个软件构件,通过组合手段高效率、高质量地构造应用软件系统的过程。由于以分布式对象为基础的构件实现技术日趋成熟,CBD 已

经成为现今软件复用实践的研究热点，被认为是最具潜力的软件工程发展方向之一。

构件的定义很多，构件在逻辑上可以看做被多个软件系统所复用的具有独立功能的系统构成成分；构件是系统的基本构造单元，具有较好的自包含性；构件可以视为一个通过接口对外界提供服务或向外界请求服务的黑盒，多个构件可以组成一个更高层次的构件，构件比“对象”提供了更高的设计抽象。

根据这个观点，可以认为构件由一方定义其规格说明，被另一方实现，然后供给第三方使用。接口（interface）是用户与构件发生交互的连接渠道，第三方只能通过构件接口的规格说明理解和复用构件，接口规格说明也是一种“契约”（contract），它足够精确地描述构件实现的功能，同时又不把构件限定于唯一的实现方法，这种不确定带来多解决方案的灵活性。另一方面，虽然构件可以独立部署，但是一个构件可能会用到其他构件或平台提供的服务，或者说在基于构件的软件系统中通常是多个构件协作完成一定功能，所以构件依赖于组装环境或称为语境（context）。

当今，构件技术已经成为计算环境的基本组成之一，众多中间件产品和开发工具提供了对不同构件模型的实现支持，特别在分布式、企业级应用软件系统中，无不把软件的构件化作为解决维护、扩展和升级的唯一途径。然而，虽然已存在了大量的 CBD 概念、方法和工具，软件业并未完全迁移到 CBD 软件开发范型，一个主要原因是缺少一套成熟的 CBD 开发方法学，另一个原因则是需要建立公共软件构件库。

公共软件构件库可提供多种构件发布、分类、检索的方法和良好的用户访问控制能力。基于构件的软件开发致力于通过组装现有软件构件的方式来建造大型软件系统，使软件构件生产成为独立的行业而存在。公共构件库管理系统的目的，是通过对可复用构件的分类、管理、存储和检索，为面向复用和基于复用的软件开发过程提供全面的支持。我国在公共软件构件库管理系统方面已经做了一些基础性的工作，为构件等软件资源的管理提供了可行的解决方案（图 2-13）。

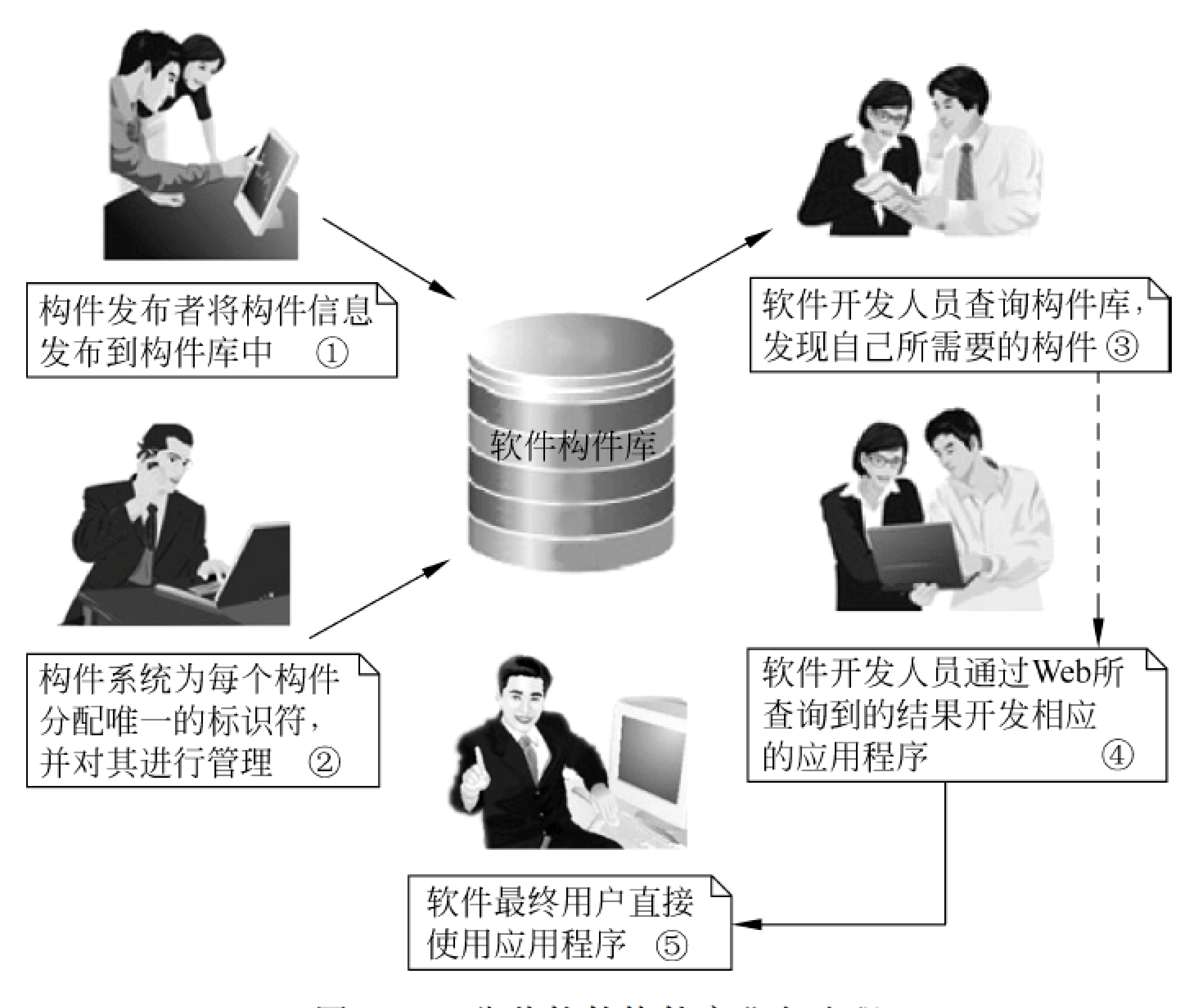


图 2-13 公共软件构件库业务流程

例如,微软公司越来越多地运用共享或通用构件来构造软件,这已经开始改变微软应用软件的结构。应用软件曾是互不相关的产品,但它们正变得越来越组合化与可以共享。举例来说,项目现在可以灵活地往不同的操作或单独的“线程执行过程”加入内含 OLE 的构件。这些构件可以向用户提供特定的功能包,如绘图、打印或数学计算。用户也可以综合来自不同产品的功能和信息对象——例如,写一封含有图形和电子表格数据的书信。

2.2.9 第 4 代技术

第 4 代技术(Fourth Generation Technique,4GT)包含了系列的软件工具,它们都有一个共同点:能使软件工程师在较高级别上说明软件的某些特征,之后工具根据开发者的说明自动生成源代码。

毫无疑问,软件能在较高的级别上被说明,就能越快地建造出程序。4GT 模型的应用关键在于说明(规约)软件的能力,或者说,它是使用一种特定的语言形式或者以一种用户可以理解的术语描述待解决问题的符号体系。

一个支持 4GT 模型的软件开发环境包含如下部分:数据库查询的非过程语言、报告生成器、数据操作、屏幕交互及定义,以及代码生成、高级图形功能、电子表格功能(图 2-14)。像其他模型一样,4GT 过程也是从需求收集这个步骤开始。理想情况下,用户应能够描述出需求,而这些需求能被转换成操作原型。因此,用户与开发者间的沟通在 4GT 方法中仍是必要的组成部分。



图 2-14 支持 4GT 模型的软件开发环境

对于较小的应用软件,可使用非过程的第 4 代语言(4GL)。4GL 基本上是面向问题的,即只需要告诉计算机“做什么”,而不必告诉计算机“怎么做”。应用 4GL 生成功能使软件开发者能够以某种方式表示期望的输出,并能够自动生成产生该输出的代码。很显然,相关信息的数据结构必须已经存在且能够被 4GL 访问。要将一个 4GT 生成的功能变成最终产品,开发者还必须进行测试,写出有意义的文档,并完成其他软件工程模型中同样要求的所有集成活动。此外,采用 4GT 开发的软件还必须考虑维护是否能够迅速实现。

4GT 模型也分别有其优点和缺点。4GT 模型的支持者认为它极大地降低了软件的开发时间,并显著提高了建造软件的生产率。反对者则认为目前的 4GT 工具并不比程序设计语言更容易使用,并且使用 4GT 开发的大型软件系统的可维护性是令人怀疑的。

2.2.10 微软公司的软件过程模型[①]

微软产品过程模型是微软公司数十年实际开发经验的精髓,微软公司的所有产品,从最初的产品策划到编程,Beta 版发行,正式版本的发布,下一个版本的开发,都遵循该

① 孙家广. 软件工程:理论、方法与实践. 北京:高等教育出版社,2008

过程模型。微软产品周期模型是整个微软开发流程的核心和基础,他们的经验值得人们关注。

微软公司的软件开发过程模型由规划、设计、开发、稳定和发布5个主要阶段组成,而且每个阶段都是由里程碑驱动的,如图2-15所示。其中,规划和设计阶段的里程碑是完成项目计划和产品特性规格说明书;开发阶段的里程碑是完成规格说明书中所列产品特性的开发;稳定阶段的里程碑是产品经过测试已达到稳定状态;发布阶段的里程碑是最终发布的产品。微软的5个阶段更像是风险驱动的、渐进的“螺旋”式的生命周期模型。这个过程模型是MSF(微软公司解决问题框架)的重要内容之一。

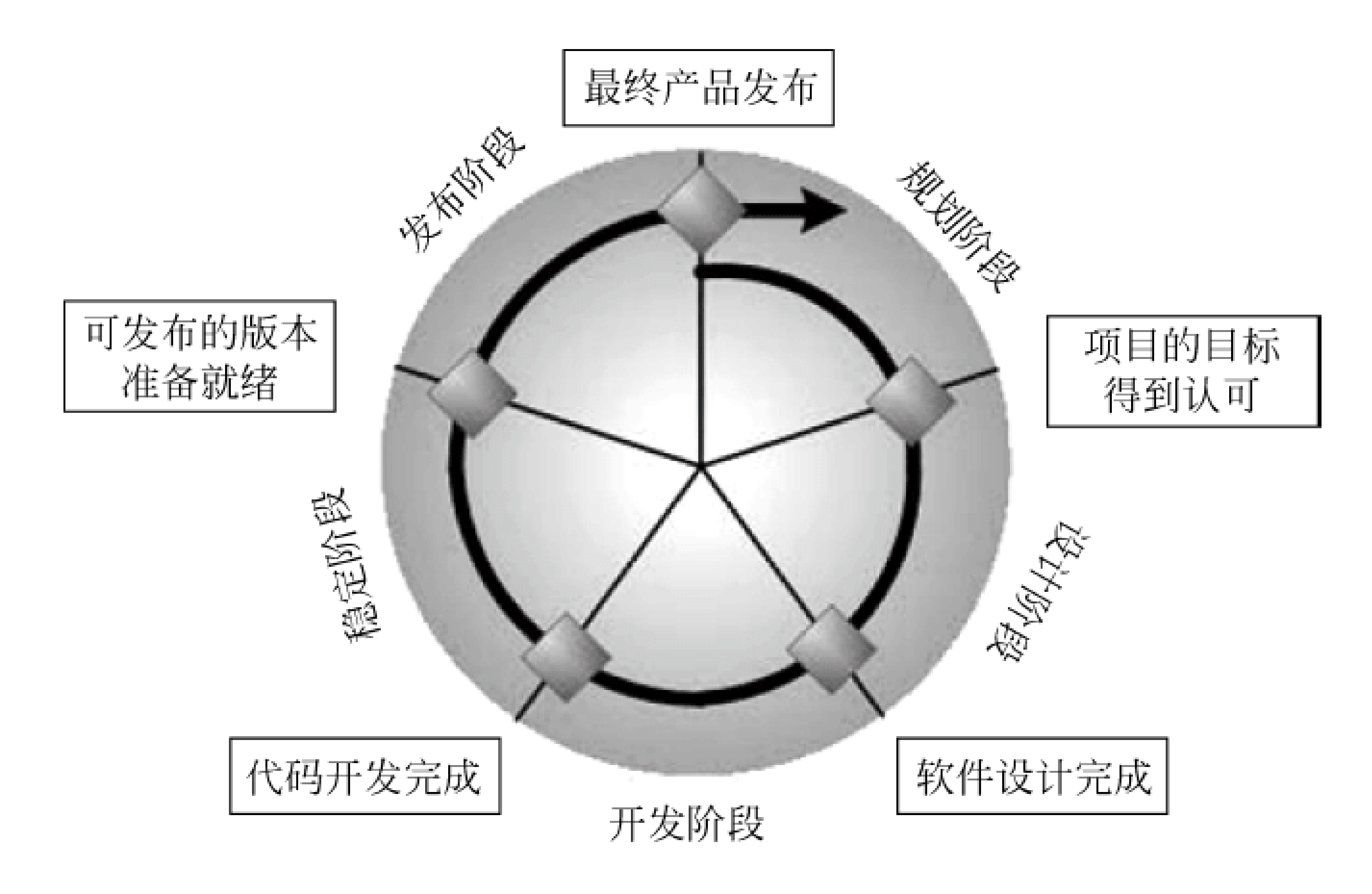


图2-15 微软公司的软件开发过程模型

(1) 规划阶段。项目团队必须对项目的前景有一个清晰的认识,明确最终要提供给客户的是什么样的产品。在这一阶段,市场经理根据市场反馈和用户需求,提出关于产品的初步构想。产品规划人员则针对产品构想开展市场调查研究,分析市场形势和自身条件,根据公司战略创建产品的市场机会文档和市场需求文档,并最终形成产品的远景目标。

(2) 设计阶段。程序经理根据产品的远景目标,完成软件的功能或特性规格说明书,并确定产品开发的主要进度。

(3) 开发阶段。开发人员根据产品功能或特性规格说明书,完成软件的开发工作。在通常情况下,为了降低软件开发的风险和管理的复杂度,开发人员往往把整个开发任务划分成若干个递进的阶段,进度表具体到每一位开发人员,并在进度表中加入缓冲的时间。此外,设置若干个内部里程碑,在每个里程碑都提交阶段性的工作成果。

(4) 稳定阶段。稳定阶段着重于对产品的测试与调试,项目在此阶段尽量不再增加新的功能。测试人员根据产品规格说明书,对开发人员提交的软件产品进行功能测试和性能测试。Bug(缺陷)管理是软件开发中非常重要的一个环节。在大型的商业软件开发中,没有Bug管理是不可想象的。在测试过程中,测试人员发现错误,并将其记录在数据库中;开发人员修正代码中的错误,测试人员再对开发人员的修正结果予以确认。无论是Windows、Office这样大型的软件,还是内部使用的各种各样的小工具,Bug的管理都贯穿于整个开发流程的始终。

（5）发布阶段。在确认产品质量符合发布标准之后，程序经理将产品的最终发布版本发送到软件生产工厂，或者直接以可下载方式发布到 Internet 上。整个软件开发工作到此结束之后，产品经理将举行产品发布会宣布产品正式上市，并通过媒体发布与产品相关的各种消息，支持工程师将提供该产品的支持服务。

如上所述，MSF（Microsoft 解决方案框架）过程模型以阶段和里程碑为基础，在某个层次上，阶段能够被简单地看做是一段时间，只不过强调了为该阶段生产相关交付内容的特定活动。但是，MSF 中的阶段要比这复杂；每个阶段都有其自身的特色，每个阶段的结束都代表了项目进展和中心点的变化。阶段可以被先后看做是探索的、调查的、创造性的、专心的和合乎规范的。里程碑是检查和同步点，用来确定阶段的目标是否已经实现。

2.3　软件过程改进

软件机构形成一套完整而成熟的软件过程不是一蹴而就的，它需要一个从无序到有序，从特殊到一般，从定性到定量，最后再从静态到动态的历程，或者说软件机构在形成成熟的软件过程之前必须经历一系列的成熟阶段。研究软件过程本质上是为了突出关键过程以改善软件的质量。人们已经得到共识，要提高软件质量必须改进软件过程。因此有必要建立一个软件过程成熟度模型来对过程做出一个客观、公正的评价，以促进软件开发组织改进软件过程。

软件过程能力描述了一个开发组织开发高质量软件产品的能力。现行的国际标准主要有两个：ISO9000.3 和 CMM。

2.3.1　软件能力成熟度模型——CMM 与 CMMI

软件过程成熟度概念的提出基于如下观点：新技术本身并不能使产品和利润增加，问题主要出现在软件过程管理上。如果按照一定的策略改进软件过程的管理，相信技术的提高则是一个自然的结果。软件过程水平整体所获得的改进将导致较高质量的软件产生，并且将会有较少的软件项目超时或超支。

软件过程成熟度是指一个特定的软件过程被显式地定义、管理、度量、控制和能行（按步骤执行）的程度。成熟度反映了软件过程能力的大小，可以用于指示企业加强其软件过程能力的潜力。当一个软件机构达到了一定的软件过程成熟级别后，它将通过制定策略、建立标准和确立机构结构使它的软件过程制度化。而制度化又促使软件机构通过建立基础设施和企业文化来支持相关的方法、实践和过程，从而使之持续并维持一个良性循环。

1. 软件能力成熟度模型

软件能力成熟度模型（Capability Maturity Model for Software Capability，SW-CMM 或 CMM）是由美国卡耐基·梅隆大学软件工程研究所（SEI）于 1987 年提出的一种软件能力评估标准，它侧重于软件开发过程的管理及工程能力的提高与评估，现已成为软件业最权威的评估认证体系。CMM 分为 5 个等级：一级为初始级，二级为可重复级，三级为已定义级，四级为已定量管理级，五级为优化级，如图 2-16 所示。

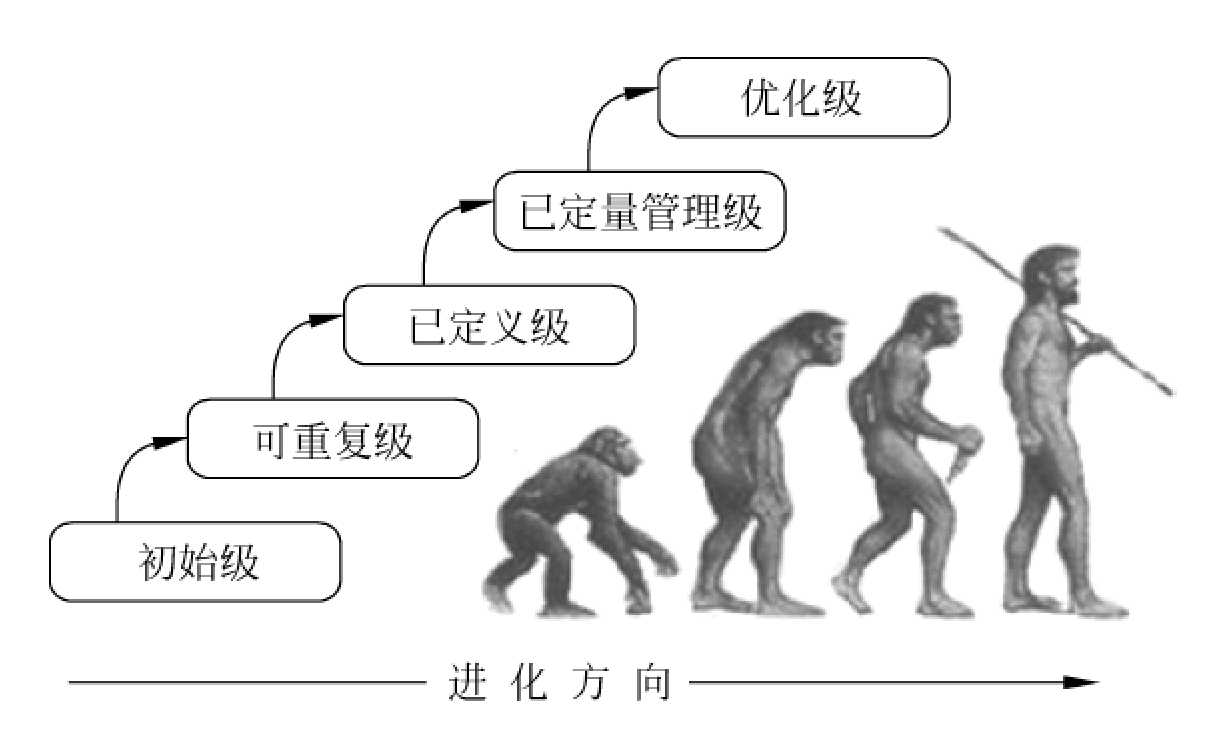


图 2-16 CMM 的 5 个级别

(1) 初始级(Initial)。处于这个最低级的组织,基本上没有健全的软件工程管理制度。软件过程是无序的,有时甚至是混乱的,对过程几乎没有定义,其软件项目的成功来源于偶尔的个人英雄主义而非群体行为,因此它不是可重复的。

(2) 可重复级(Repeatable)。在这一级,软件机构的项目计划和跟踪稳定,项目过程可控。这个级别使用了基本的软件项目管理措施,根据从类似产品中获得的经验对新的产品进行计划和管理。因而,这个级别的管理是可重复的。

(3) 已定义级(Defined)。在这一级,软件机构已为软件生产的过程编制了完整的文档。软件过程的管理方面和技术方面都明确地做了定义,并按需要不断地改进过程,而且采用评审的办法来保证软件的质量。软件过程已被提升成标准化过程,从而更加具有稳定性、可重复性和可控性。

(4) 已定量管理级(Managed)。软件过程和软件产品都有定量的目标,并被定量地管理,因而其软件过程能力是可预测的,其生产的软件产品是高质量的。

(5) 优化级(Optimizing)。其特点是过程的量化反馈和先进的新思想、新技术促进过程不断改进,技术和过程的改进被作为常规的业务活动加以计划和管理。

除了初始级外,每一个成熟度等级又由若干个关键过程区域(Key Process Areas)构成。关键过程区域指出为了达到某个成熟度等级所要着手解决的问题。达到一个成熟度等级,必须实现该等级上的全部关键过程区域。要实现一个关键过程区域,就必须达到该关键过程区域的所有目标。

CMM2 级过程区域有 7 个:需求管理、项目策划、项目监督和控制、供方协定管理、测量和分析、过程和产品质量保证、配置管理。

CMM3 级过程区域有 11 个:需求开发、技术解决、产品集成、验证、确认、组织过程聚焦、组织过程定义、组织培训、集成项目管理、风险管理以及决策分析和决定。

CMM4 级过程区域有 2 个:组织过程性能和定量项目管理。

CMM5 级过程区域有 2 个:组织革新和部署、原因分析和决定。

当一个软件组织按照 CMM 的要求贯彻活动,并达到预期的效果,该组织就可以被认为是达到了 CMM 的要求。

CMM 是科学评价一个软件企业开发能力的标准,但要达到较高的级别也非常困难,根据 1995 年美国所做的软件产业成熟度的调查,在美国的软件产业中,CMM 成熟度等级为

初始级的占 70%，为可重复级的占 15%，为定义级的所占比例小于 10%，为管理级的所占比例小于 5%，为优化级的所占比例小于 1%。

2. CMM 的内部结构

CMM 为软件过程能力的提高提供了一条改进的途径，每个成熟度等级有着各自的功能。除第一级外，CMM 的每一级按完全相同的内部结构构成，如图 2-17 所示。

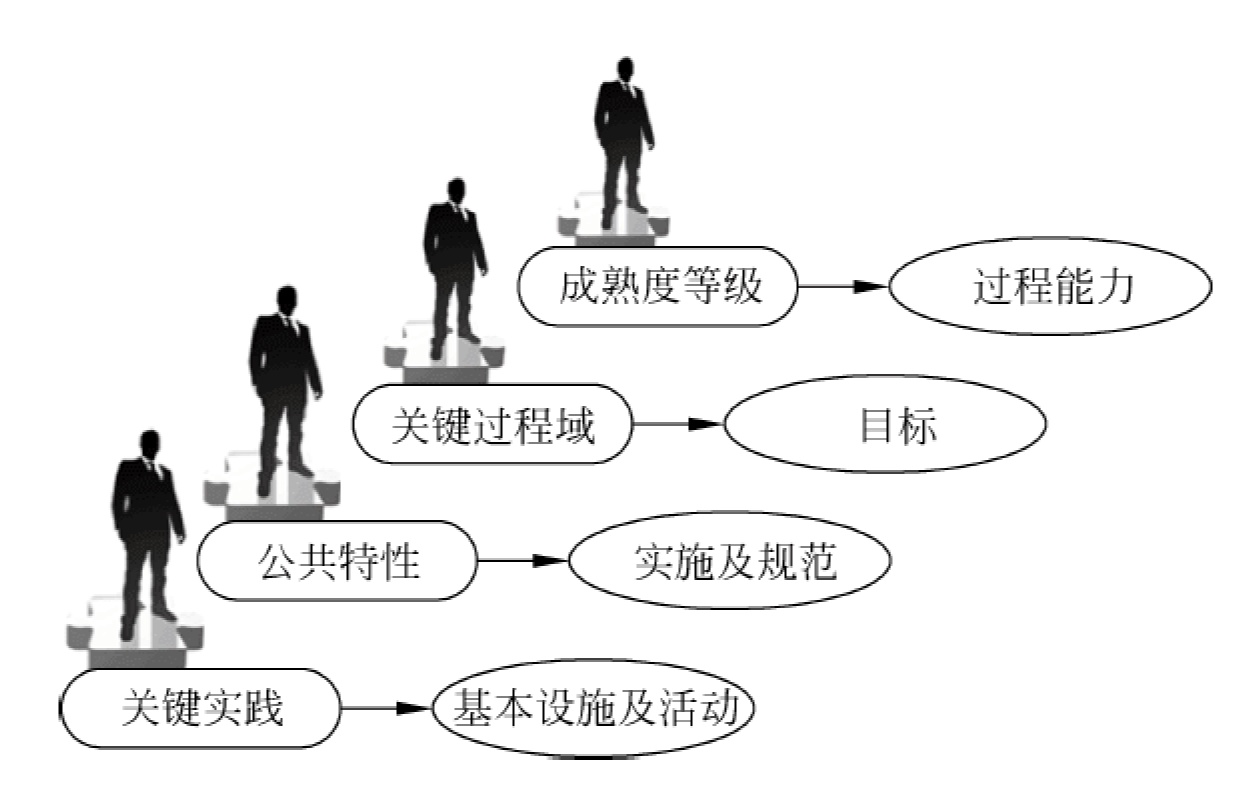


图 2-17 CMM 的内部结构图

(1) 成熟度等级。每个成熟度等级都是在朝着实现成熟软件过程进化途中的一个妥善定义的平台。5 个成熟度等级就如同 5 个台阶，每达到一个成熟度等级就如同登上了一个台阶，标志着软件过程达到了一个质的飞跃。这 5 个成熟度等级是 CMM 的顶层结构。

(2) 过程能力。这里的过程能力特指软件过程能力，软件过程能力描述通过遵循软件过程能实现预期结果的程度。一个组织的软件过程能力为我们提供了一种方法，通过这种方法可以预测该组织承担下一个软件项目所产生的结果。可见，软件过程能力关注的是预期的结果，而不是实际的结果。软件过程性能表示的才是遵循软件过程所得到的实际结果。

(3) 关键过程域。每个成熟度等级由若干关键过程域组成。每个关键过程域标识出一串相关的活动，当这些活动作为群体完成时，就实现了一组目标，此组目标对于达到该成熟度等级是至关重要的。关键过程域分别定义在各个成熟度等级之内，并与其所在的成熟度等级紧密相连。

(4) 关键实践。每个关键过程域用若干关键实践描述，实施这些关键实践，能帮助实现该关键过程域的目标。关键实践描述对关键过程域的有效实施和规范化贡献最大的基础设施和活动。

3. 能力成熟度模型集成——CMMI

软件能力成熟度模型 CMM 取得了成功，产生了很大影响，已经得到了国际软件产业界的认可，成为当今(企业)从事规模软件生产不可缺少的一项内容。在 CMM 公布后的若干年内工程环境更加复杂，工程规模更大、参与工程项目的组织和人员更多、范围更广泛，工程的施工涉及多学科、交叉学科、并行工程及更多的国际标准。这些新的变化促使美国国防部、美国国防工业协会和 SEI/CMU 共同开发一种新的模型——CMMI(Capability Maturity Model Integration，能力成熟度模型集成)。CMMI 项目在 1998 年正式启动，来自

业界、政府部门和 SEI/CMU 这 3 个方面的上百位研究者，经过两年的工作于 2000 年发布 CMMI V1.0 版本。

CMMI 为改进一个组织的各种过程提供了一个单一的集成化框架，新的集成模型框架消除了各个模型的不一致性，减少了模型间的重复，增加了透明度和理解，建立了一个自动的、可扩展的框架，因而能够从总体上改进组织的质量和效率。CMMI 内容分为"要求"、"期望"和"提供信息"3 个级别，来衡量模型包括的质量重要性和作用。最重要的是"要求"级别，它是模型和过程改进的基础。第二级别"期望"在过程改进中起到主要作用，但是某些情况不是必需的，可能不会出现在成功的组织模型中。"提供信息"构成了模型的主要部分，为过程改进提供了有用的指导，在许多情况下它们对需要和期望的构件做了进一步说明。

CMMI 提供了阶段式和连续式两种表示方法，但是这两种表示法在逻辑上是等价的。阶段式模型基本沿袭 CMM 模型框架，仍保持 5 个"成熟度等级"，但对过程域做了一些调整和扩充，当某一组织通过了某一等级过程域中的全部过程，即意味着该组织的成熟度达到了这一等级。利用阶段式模型对组织进行成熟度度量，概念清晰、易于理解、便于操作。连续式模型的过程域强调实践，每个过程域代表组织某一方面的能力。每个过程域采用 0～5 分的记分法则，过程域成熟度的评估比较准确，能反映过程域的实际情况，特别是优势和不足。所有过程域共同的能力等级决定组织的能力等级。连续式模型允许组织对连续式模型的过程域进行剪裁，也允许对不同的过程域采用不同的能力等级。

2.3.2 CMM/CMMI 的应用及面临的问题

CMM/CMMI 目前代表着软件发展的一种思路，一种提高软件过程能力的途径，为软件行业的发展提供了一个良好的框架，而且是改进软件过程能力的有用工具。CMM/CMMI 是提高组织的成熟度和软件过程能力的有效模型和工具，几千个组织的多年实践表明，CMM 是成功的、有效的。一个软件组织无论是采用 CMM 模型还是使用 CMMI 模型，无论是使用阶段式模型还是使用连续式模型都能提高组织的成熟度，都能提高项目的软件过程能力，用两种模型或两种方法评价得到的结论应该是基本一致的。

随着 CMM/CMMI 应用得不断深入和广泛，一些问题也显现出来。这些问题是在应用 CMM 时必须要面对和解决的问题，否则就无法适当地应用 CMM。

(1) CMM 主要面向大型项目，其体系比较庞大，实施起来成本昂贵。因此，对于较小的组织来说，根本无力承担应用和实施 CMM 的成本。即使对于大型企业组织来说，也不可能照搬 CMM。如何对 CMM 裁剪，构造轻型和小型的 CMM，以适应不同的企业和组织的需要，是当前研究的重点。

(2) CMM 的变更控制十分严格，要求一开始的时候就尽可能地将需求固定下来。这在一定程度上造成了开发过程的僵化，对于当前软件业来说也是很难实现的。如何增强 CMM 的灵活度，敏捷方法提供了很好的思路。CMMI 是两种结构、三种模型的综合，在这些结构和模型尚处于发展，还不十分成熟的情况下，CMMI 模型由于照顾各方面的意见显得复杂、臃肿，给使用和评估带来了困难。

(3) CMM/CMMI 是基于过程进行管理的，它指出了 what to do，但是没有指出 how to do，可操作性比较差。如何增强 CMM 的可操作性是当前研究的热点。SEI 也已经认识到 CMM 在可操作性上的不足，继而又提出了团队软件过程(Team Software Process，TSP)和

个体软件过程(Personal Software Process,PSP),它们是面向开发小组和开发个人对 CMM 进行的微观优化,对 CMM 的实施提供了一些指导和帮助。

2.3.3　个体软件过程

个体软件过程(PSP)是由美国卡耐基·梅隆大学软件工程研究所(CMU/SEI)领导开发的,于 1995 年推出,在软件工程界引起了极大的轰动,可以说是由定向软件工程走向定量软件工程的一个标志。

PSP 是一种可用于控制、管理和改进个人工作方式的自我改善过程,是一个包括软件开发表格、指南和过程的结构化框架。据统计,软件项目开发成本的 70% 取决于软件开发人员个人的技能、经验和工作习惯。PSP 能够指导软件开发人员保证自己的工作质量,估计和规划自身的工作、度量和追踪个人表现,管理自身的软件过程和产品质量。

PSP 与具体的技术(程序设计语言、工具或者设计方法)相对独立,其原则能够应用到几乎任何的软件工程任务之中。PSP 能够说明个体软件过程的原则;帮助软件工程师做出准确的计划;确定软件工程师为改善产品质量要采取的步骤;建立度量个体软件过程改善的基准;确定过程的改变对软件工程师能力的影响。

CMM 侧重于软件企业中有关软件过程的宏观管理,面向软件开发单位。PSP 则侧重于企业中有关软件过程的微观优化,面向软件开发人员。二者互相支持,互相补充,缺一不可。就像 CMM 为软件企业的能力提供一个阶梯式的进化框架一样,PSP 为个体的能力也提供了一个阶梯式的进化框架(图 2-18),由 4 级组成,每一级都试图指出过程缺陷并提供解决方法。PSP 以循序渐进的方法介绍过程的概念,每一级别都包含了更低一级别中的所有元素,并增加了新的元素。这个进化框架是学习 PSP 过程基本概念的好方法,它赋予软件人员度量和分析工具,使其清楚地认识到自己的表现和潜力,从而可以提高自己的技能和水平。



图 2-18　PSP 进化模型

(1) 个体度量过程 PSP0。学会通过表格采集项目开发活动中的各种数据,建立度量基线。记录问题和具体分析解决问题的措施,提高个体的质量意识和过程意识。

(2) 个体规划过程 PSP1。引入了基于估算的计划方法,用自己的历史数据来预测新程序的大小和需要的开发时间。

(3) 个体质量管理过程 PSP2。建立自己程序的质量目标,根据程序允许的缺陷率来建立检测表,按照检测表进行设计复查和代码走查,将流入到集成和系统测试阶段的缺陷控制在一定范围内。

(4) 个体循环过程 PSP3。把个体开发小程序所能达到的生产效率和生产质量,延伸到大型程序;其方法是采用螺旋式上升过程,即迭代增量式开发方法。首先把大型程序分解成小的模块,然后对每个模块按照 PSP2 所描述的过程进行开发,最后把这些模块逐步集成为完整的软件产品。应用 PSP3 开发大型软件系统,必须采用增量式开发方法,并要求每一个增量都具有很高的质量。

2.3.4 团队软件过程

1. 什么是 TSP

团队软件过程(TSP)可以为开发软件产品的开发团队提供指导,它的早期实践侧重于帮助开发团队改善其质量和生产率,以使其更好地满足成本及进度的目标。TSP 被设计为满足 2～20 人规模的开发团队,大型的多团队过程的 TSP 被设计为大约 150 人的规模。

TSP 加上 PSP 帮助高绩效的工程师在一个团队中工作,来开发有质量保证的软件产品,生产安全的软件产品,改进组织中的过程管理。通过 TSP,一个组织能够建立起自我管理的团队来追踪他们的工作、建立目标,并拥有自己的过程和计划(图 2-19)。这些团队可以是纯粹的软件开发团队,也可以是集成产品的团队。TSP 团队在广泛领域里可能运用 XP(极限编程)、RUP 或其他方法。TSP 使具备 PSP 的工程人员组成的团队能够学习并取得成功。如果软件组织运用 TSP,它会帮助软件组织建立一套成熟规范的工程实践,确保软件安全可靠。



图 2-19 团队软件过程

2. 实施 TSP 的前提

实施 TSP 的先决条件有以下 4 条:

(1) 必须有高层主管和各级经理的支持,以取得必要的资源,这一点非常重要。

(2) 项目组开发人员需要经过 PSP 的培训并有按 TSP 工作的愿望和热情。开发人员必须在参与 TSP 团队构建或执行 TSP 过程以前获得这些技能的培训。

(3) 整个开发小组在总体上应处于 CMM 2 级以上。

(4) 开发小组成员应在 2～20 个人之间。他们为一个共同的目标或任务而工作,每个人都被分配了一定的角色或职责。

PSP 培训包括学习如何编制详细的计划,采集和使用过程数据,用获得的数据跟踪项目,度量和管理产品质量以及定义和使用可操作的过程。

3. TSP 的设计原则

设计一个开发过程有许多方法,在 TSP 中,有以下一些主要设计原则。

(1) 循序渐进的原则。首先在 PSP 的基础上提出一个简单的过程框架,然后逐步完善。

(2) 迭代开发的原则。选用增量式迭代开发方法,通过几个循环开发一个产品。

(3) 质量优先的原则。对按 TSP 开发的软件产品,建立质量和性能的度量标准。

(4) 定期评审的原则。在 TSP 的实施过程中,对角色和团队进行定期的评价。

(5) 过程规范的原则。对每一个项目的 TSP 规定明确的过程规范。

(6) 指令明确的原则。对实施 TSP 中可能遇到的问题提供解决问题的指南。

2.3.5 CMM、TSP、PSP 三者的关系

CMM,TSP 和 PSP 为软件产业提供了一个集成化的、三维的软件过程改革框架。三者互相配合,各有侧重,形成了不可分割的整体,缺一不可。在 CMM 的 18 个关键过程域中,有 12 个与 PSP 紧密相关,有 16 个与 TSP 紧密相关。因此,如果能够熟悉个体软件过程和团队软件过程,不仅有助于工程师改善工作效率,而且也非常有利于组织的过程改善。

在软件过程改进中,CMM、TSP 和 PSP 3 者的关系如图 2-20 所示。

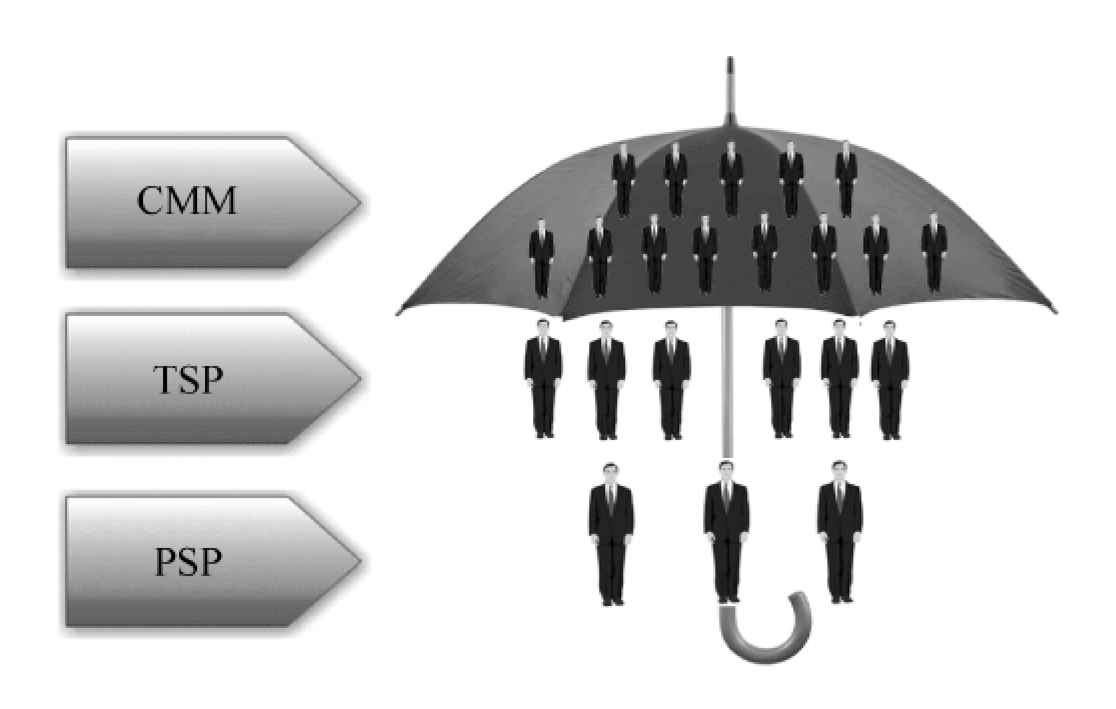


图 2-20 CMM、TSP 和 PSP 三者的关系

CMM 是过程改进的第一步,它注重于组织能力和高质量的产品,提供了评价组织的能力、识别优先改善需求和追踪改善进展的管理方式。

TSP 注重团队的高效工作和产品交付能力,结合 PSP 的工程技能,使软件工程师知道如何将个体过程结合进团队软件过程,使管理者懂得如何支持和授权项目团队,并且依据规范的工程实践进行项目的管理,以生产高质量的产品。

PSP 注重个人的技能,能够指导软件工程师如何保证自己的工作质量,估计和规划自身的工作,度量和追踪个人的表现,管理自身的软件过程和产品质量。经过 PSP 学习和实践的正规训练,软件工程师们能够在他们参与的项目工作之中充分利用 PSP,从而保证了项目整体的进度和质量。

总之,单纯实施 CMM,并不能真正做到软件组织成熟度的升级,只有将实施 CMM 与实施 TSP 和 PSP 有机地结合起来,才能发挥最大的效力。因此,软件过程改进框架应该是 CMM,TSP 和 PSP 的有机集成。

本 章 小 结

软件工程过程是生产一个最终能满足需求且达到工程目标的软件产品所需要的步骤。为了有效指导软件项目的开发,可使用软件开发模型来清晰、直观地表达软件开发全过程。过程定义了方法使用的顺序、可交付产品(文档、报告以及格式等)的要求、帮助确保质量和变更的控制,使软件管理人员能对它们的进展进行评价。

软件开发过程是随着开发技术的演化而随之改进的。从早期的瀑布式开发模型到后来出现的螺旋式的迭代开发,以及最近开始兴起的敏捷开发方法,它们展示出了在不同的时代

软件产业对于开发过程的不同的认识，以及对于不同类型项目的理解方法。

本章根据不同软件开发的特点和需求，分别介绍了几种典型的软件过程模型，用户可根据实际开发需要进行选择。

注意区分软件开发过程和软件过程改进之间的重要区别。例如，像 ISO 9000、CMM、TSP、PSP 这样的名词阐述的是一些软件过程改进框架，它们提供了一系列的标准和策略来指导软件组织如何提升软件开发过程的质量、软件组织的能力，而不是给出具体的开发过程的定义。

思考与练习

1. 什么是过程？过程有哪些特征？

2. 什么是软件过程？软件过程的公共框架的内容是什么？

3. 常用的软件过程有哪些？

4. 演化软件过程模型包括增量模型和螺旋模型，各举出一个可以采用增量模型和螺旋模型的特定的软件项目，并给出在软件中应用该模型的某个场景。

5. 快速原型开发方法的基本思想与设计步骤是什么？试给出 1～2 个可以采用原型方法的软件开发项目的实例，并举出一个或两个难以使用原型的应用。

6. 试根据统一软件过程(RUP)二维开发模型，简述 RUP 的生命周期的阶段与工作流的含义。

7. 什么是构件？基于构件的开发(CBD)方法为何是具有潜力的软件工程发展方向？

8. 微软公司的软件开发过程模型由哪几个主要阶段组成？

9. 你认为本章给出的哪一个软件工程模型最有效？为什么？

10. 什么是软件过程成熟度？CMM 的 5 个等级是如何划分的？

11. 在 CMM 的 18 个关键过程域中，有 12 个与 PSP 紧密相关，通过上网查阅相关资料，比较 CMM 与 PSP 各自关注的过程域有何异同。

12. 什么是 TSP？实施 TSP 的前提条件是什么？

13. 简述 CMM、TSP、PSP 3 者之间的关系。

第3章 软件工程领域下的项目管理

项目管理就是把各种知识、技能、手段和技术应用于项目活动之中，以达到项目的要求。

——项目管理知识体系指南(第3版)

人类的活动可以分为两大类：一类是重复性、连续不断的活动，如人类的各种周而复始的生产性活动(如农耕)；另一类是独特的活动，称为"项目"。项目是为提供某项独特产品、服务或成果所做的一次性工作任务。由于人类社会的大部分活动都可以按项目来运作，因此当代的项目管理已深入到各行各业，以不同的类型，不同的规模出现。例如，小到一次会议策划与实施、一个小型应用软件的开发，大到建设三峡水利工程、"神舟"载人航天工程等。因此，Everything is Project(事事皆项目)，项目管理与人类社会的发展息息相关。

在相当长的一段时期内，应用项目管理的主要是国防建设和建筑工程。时至今日，项目管理迅速发展到计算机工程、电子通信、航天工程、金融业甚至政府机关等众多领域。目前，许多国家政府、企业、社会团体以及国际组织都在应用"项目管理"进行有效的开发工作。

3.1 项目管理的历史实践

管理是无边界的大概念，任何事物都需要管理。从某种意义上说，管理是使事物的发展从混乱无序走向有序有效发展的唯一方法。管理与人类发展并存，人类从原始走向现代，管理也从低级走向高级，从自发走向自觉，从分散孤立的思想和方法，走向综合统一的学科体系。

从根本上讲，项目管理并不神秘，人类数千年来进行的组织工作和团队活动，都可以视为项目管理行为。从古埃及金字塔到延绵万里的中国长城，都是古代文明创造的伟大工程实践的成功范例。人类进步的文明史由此改写。

3.1.1 远古的伟大工程实践

"愚公移山"是《汤问篇》中记载的一个妇孺皆知的故事，寓意深刻。如果抛开故事的神话色彩，在这个故事中可以看到古人对"移山"工程的基本描述。

首先，我们看到了原始需求的产生："惩山北之塞，出入之迂"；我们也看到了项目沟通的基本方式："聚室而谋曰"；然后，我们看到愚公确定了一个项目的目标："毕力平险，指通豫南，达于汉阴"，并通过研讨，择定了一个井然有序的、可以实现的技术方案："扣石垦壤，箕畚运于渤海之尾。"

在这个项目中，动用了3名技术人员和1名工程管理人员："(愚公)率子孙荷担者三夫"，并获得了1名力量较弱，但满富工作激情的外协："邻人京城氏之孀妻，有遗男，跳往助之"。至此，已经描述了"愚公移山"整个工程的项目概况。当然，该工程的最后结局是完美的，愚公的精神感动了天帝，命令天神把两座大山搬走了。

如果说"愚公移山"只是一个神话故事，那么，中国的长城则是人类文明史上最伟大的建筑工程，它始建于2000多年前的春秋战国时期，秦朝统一中国之后连成万里长城。汉、明两代又曾大规模修筑。其工程之浩繁，气势之雄伟，堪称世界奇迹。万里长城是世界上修建时间最长，工程量最大的，是冷兵器战争时代的国家军事性防御工程，凝聚着我们祖先的血汗和智慧。即使在工程技术发达的今天，长城的规模与工程量之大也是令人叹为观止的。

这两则故事尽管文化背景不同，但是在很多方面和不同层次上都是非常深刻和富有教育意义的。让我们将它仅仅作为纯粹的工程项目，来看看在管理上有什么值得学习的方面。即使按照现代工程项目的标准，这两个项目还是有许多条件符合的。

(1) 项目的原始需求？建造长城的原始需求是"寇扼于墙堑，散漫不得出"[①]。虽然"愚公移山"的原始需求显得很幼稚，但动机明确。

(2) 项目的目标？目标十分清晰，如"筑长城亭障，万里安边"[②]。

(3) 项目资源？"乃使蒙恬将三十万众北逐戎狄，收河南"，可见人力非常充足，且土石材料也取之不尽，用之不完。

(4) 时间与进度？没有任何时间限制的迹象，项目甚至可以跨越千年而继续。

(5) 足够的技术？采用的技术尽管原始，但在那个远古时代却是可行的。

(6) 项目管理？遵循"筑长城，因地形，用制险塞"[③]的原则。

(7) 项目风险？河曲智叟实际看到了愚公移山的项目风险，其笑曰："甚矣，汝之不惠。以残年余力，曾不能毁山之一毛，其如土石何？"

3.1.2 沟通的故事——巴比伦塔的倒塌

另一个故事是发生在西方的创世纪之初，据《创世纪》记载：当时人类子嗣繁衍，建设了一座大城——巴比伦城，人们要在城中建造一座高塔，高耸入云，几乎要触及天堂的高度了。巴比塔的建造触怒了上帝，便使建造高塔的人们彼此说起不同的语言，这样人们就不能顺利沟通，巴比伦塔也因此而倒塌了。巴比伦塔是人类继诺亚方舟之后的第二大工程壮举，同时也是第一个彻底失败的工程。

巴比伦塔给我们的管理教训就是它们缺乏沟通和交流，以及交流的结果——组织。他们无法相互交谈，从而无法合作。当合作无法进行时，工作陷入了停顿。通过史书的字里行间，我们推测交流的缺乏导致了争辩、沮丧和群体猜忌。很快，部落开始分裂，大家选择了孤立，而不是互相合作。

不仅在创世纪之初，即使在文明充分发展的现代，沟通仍是整个项目团队的核心要素，关于项目的目标、进度任务、问题、风险、思想等都需要通过沟通来传达。有效的沟通是项目

---

① 明史·余子俊传. 中华书局，1974，178：4738

② 汉·司马迁. 史记

③ 汉·司马迁. 史记

成功必不可少的要素。

3.2 软件项目管理的范围与内容

项目和项目管理是在一个大于项目本身的环境中进行的。项目管理团队必须理解这个大于项目的环境,只有这样才能选择适合项目的软件过程、方法和工具。

3.2.1 什么是项目管理

尽管人类的项目实践可以追溯到几千年前,但是将项目管理作为一门科学来进行分析研究的历史并不长。人们通常认为,项目管理是第二次世界大战的产物,起始于曼哈顿项目(Manhattan Project),它是美国陆军部于 1942 年 6 月开始实施的利用核裂变反应来研制原子弹的计划。为了先于纳粹德国制造出原子弹,在工程执行过程中,项目管理者应用了系统工程的思路和方法,大大缩短了工程所耗时间。这一工程的成功促进了第二次世界大战后系统工程的发展和一个新的管理方法——"项目管理"的诞生。项目管理就是为了实现一个确定的目标,从开始创意到结果完成,进行全过程全方位全要素的管理。

自 20 世纪 60 年代以来,学术界与各有关专业人士对项目管理的研究集中在两个大的方面:一方面是各领域的专家们在探讨如何将本学科领域的专业理论及方法应用于项目管理,如计算机、控制论、模糊数学等;另一方面则是各行各业的专家们在研究如何把项目管理的理论及方法应用到本行业中去,如建筑业、农业、军事工业、软件行业等。

根据美国项目管理学会(PMI)对项目管理的定义:"项目管理就是把各种知识、技能、手段和技术应用于项目活动之中,以达到项目的要求。项目管理是通过应用和综合诸如启动、规划、实施、监控和收尾等项目管理过程来进行的。"[①]PMI 开发了一套项目管理知识体系(Project Management Body of Knowledge,PMBOK)。该知识体系把项目管理分为以下 9 个知识领域(如图 3-1 所示)。

(1) 项目集成管理(Project Integration Management)。

(2) 项目范围管理(Project Scope Management)。

(3) 项目时间管理(Project Time Management)。

(4) 项目成本管理(Project Cost Management)。

(5) 项目质量管理(Project Quality Management)。

(6) 项目人力资源管理(Project Human Resource Management)。

(7) 项目沟通管理(Project Communications Management)。

(8) 项目风险管理(Project Risk Management)。

(9) 项目采购管理(Project Procurement Management)。

每一个知识领域都包括若干个子域,例如,项目范围管理包括的子域有范围规划、范围定义、制定工作分解结构、范围核实和范围控制。

---

① 项目管理知识体系指南. 第 3 版

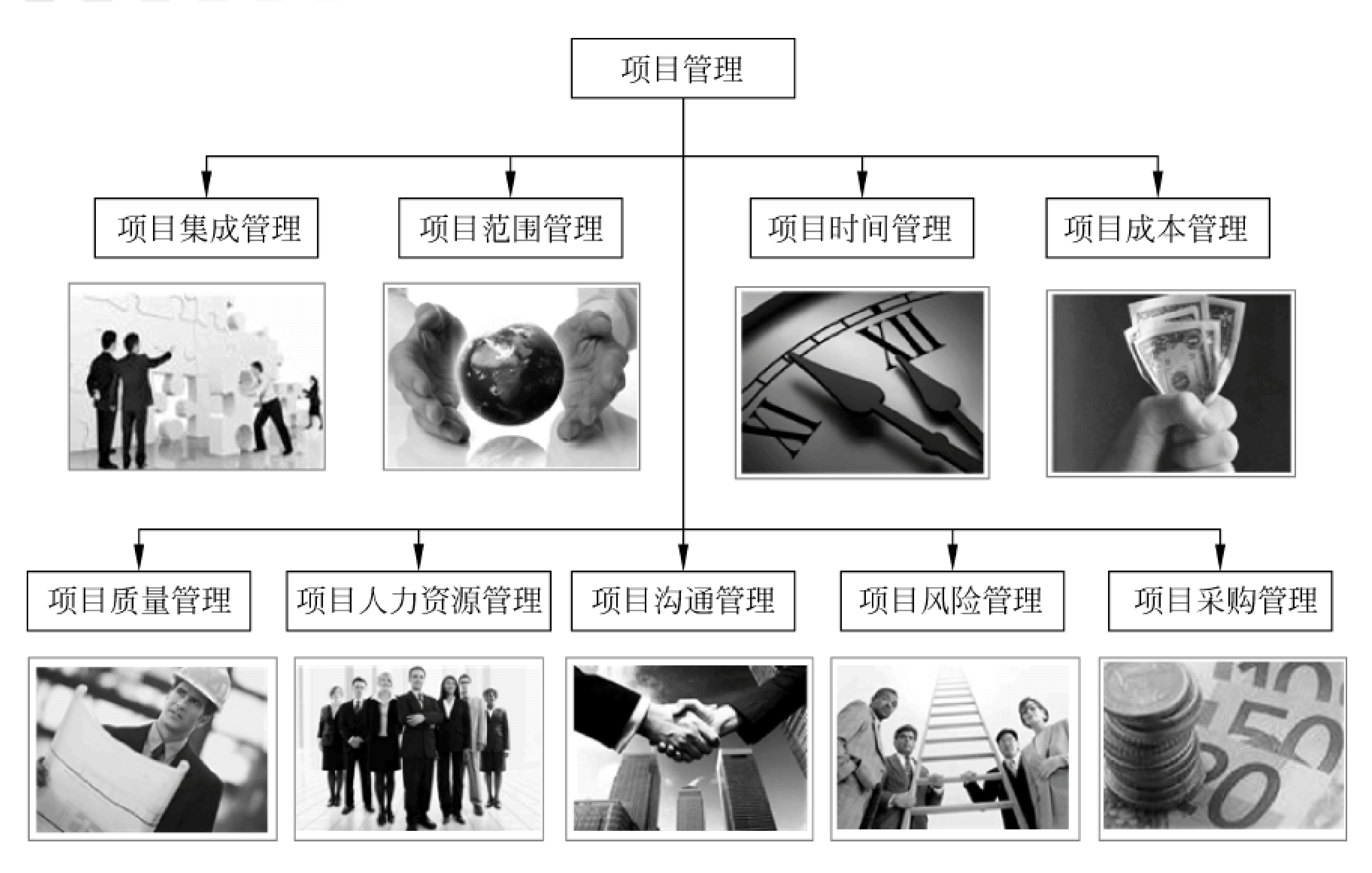


图 3-1　PMI 的项目管理知识体系

3.2.2　软件项目管理的范围

项目范围管理是确保项目包括成功完成项目所需的全部工作,但又只包括必须完成的工作的各个过程。它主要关心的是确定与控制那些应该与不应该包括在项目之内的过程[①]。

软件项目是以软件为产品的项目。软件产品的性质决定了软件项目管理和其他领域的项目管理不同。首先,软件是纯知识产品,其开发进度和质量很难估计和度量,生产效率也难以预测和保证。其次,软件系统的复杂性也导致了开发过程中各种风险的难以预见和控制。Windows 这样的操作系统有数千万行以上的代码,同时有数千个程序员在进行开发,项目经理都有上百个。这样庞大的系统如果没有很好的管理,其软件质量是难以想象的。

软件项目管理的根本目的是为了让软件项目尤其是大型项目的整个软件生命周期(从分析、设计、编码到测试、维护全过程)都能在管理者的控制之下,以预定成本按期、按质地完成软件并交付用户使用。软件项目管理先于任何技术活动之前开始,并且贯穿于软件的整个生命周期。

软件项目管理就是为了使软件项目能够按照预定的成本、进度、质量顺利完成,而对人员(People)、产品(Product)、过程(Process)和项目(Project)进行分析和管理的活动,这 4 个要素构成了软件项目管理的主要内容(图 3-2)。

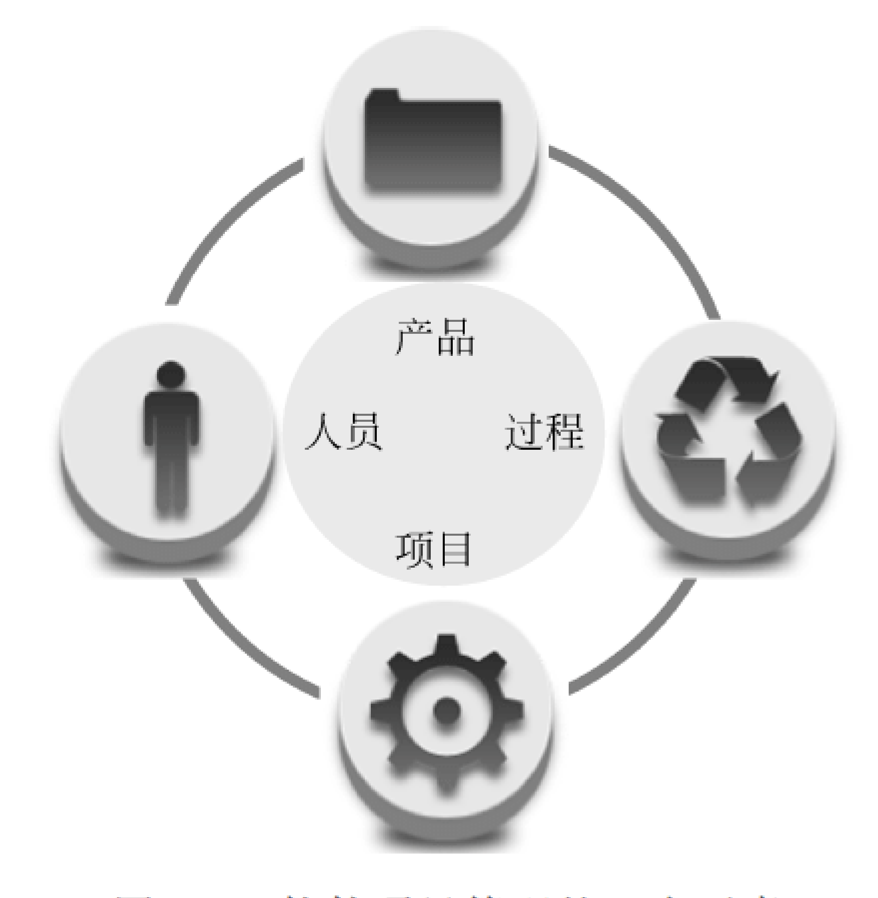


图 3-2　软件项目管理的 4 个要素

① Turner, Rodney J. 项目基础管理手册. McGraw-Hill, 1992

3.2.3　人员

在构建软件的过程中，人员因素起直观作用，这是基本的事实。工具很重要，技术很重要，过程当然也很重要，但是人员的作用远远超过其他因素。关于人员重要性的最早表述有："影响软件可靠性的首要因素是寻找、激励和管理软件的设计和维护人员"；其他简洁的表述有："人员是成功的关键"，"揭开各种工程方法的表面，寻找成功的原因，答案是人员"，"说到底，决定软件生产效率的最终因素是每个参与者的能力"①。

1. 软件项目的人员结构

任何管理者如果忘记了软件工程是人的智力密集的劳动，他就永远不可能在项目管理上得到成功；软件产品是人们大量智力劳动的结晶，软件项目能否获得成功，人的因素在其中所起的作用比其他任何工程项目都突出。用户是否参与及软件人员的能力等人的因素对软件生产率的影响极大。在与软件成本相关的影响因素中，人员的能力是最大影响因素。如果在软件项目中能够充分发挥软件人员的积极性，使他们的才能得到尽量的施展，软件生产率可大为提高。

软件工程专家 Tom DeMarco 积 30 年软件项目管理的经验，认为软件项目中对于人员的管理问题不能像其他事物那样简单地划分，机械地对待。他特别注意到项目的规模、成本、缺陷、加快开发的因素以及执行进度计划中的种种问题。积累了 500 个项目开发过程的数据，从中发现大约 15%的项目失败了。有的是一开始就被撤销，有的中途流产，有的推迟了进度，有的成果不能投入使用。而且项目规模越大，情况越糟。究其原因，绝大多数失败的项目竟找不出一个可以说得出口的技术障碍；而障碍却来自人员之间的联系问题、人员的任用问题、对上级或对雇主失望、工作缺乏动力或缺乏高额工程维持费用等。这些人际关系问题的解决可归结于"软件项目社会学"。

参与软件过程(及每一个软件项目)的人员可以分为以下 5 类。

(1) 高级管理者。负责确定商业问题，这些问题往往对项目产生很大影响。

(2) 项目(技术)管理者。必须计划、刺激、组织和控制软件开发人员。

(3) 开发人员。负责开发一个产品或应用软件所需的专门技术人员。

(4) 客户。负责说明待开发软件的需求的人员。

(5) 最终用户。一旦软件发布成为产品，最终用户是直接与软件进行交互的人。

每一个软件项目都有上述的人员参与。为了获得很高的效率，项目组必须最大限度地发挥每个人的技术和能力，这是项目管理者的任务。

2. 开发团队与个体

团队是由一定数量的个体组成的集合，这个团队包括以上所述的 5 类人员及其他利益相关者。通过将具有不同潜质的人组合在一起，形成一个具有高效团队精神的队伍来进行软件项目的开发。团队开发是发掘作为个体的个人能力，然后发掘作为团队的集体能力。

① Glass. 软件工程的事实与谬误. 严亚军，龚波，译. 北京：中国电力出版社，2005

当一组人称为团队时，他们应该承诺为一个共同的目标工作，每个人的努力必须协调一致，而且能够愉快地在一起合作，从而开发出高质量的软件产品。

一个良好和管理有序的开发团队，必须明确列出团队中每个成员（个体）完成项目所需的角色和职责，需考虑下述各项内容。

（1）角色：指某人负责的项目的某部分工作的标识。角色的明确性（包括职权、责任和边界）对于项目的成功至关重要。例如，微软公司的项目组中包含6种最重要的工作角色，分别是产品管理角色、程序管理角色、开发角色、测试角色、用户体验角色和发布管理角色，如图3-3所示。这6种工作角色处于对等的项目组结构中，各自完成特定的职能。在6种角色构成的环形结构里，"沟通"占据了核心地位。这是因为在团队模型中，交流和沟通是促使各种角色协同工作，共同完成项目目标的关键因素。



图3-3 团队模型的6种角色

（2）职权：指使用项目资源及做出决策和批准的权力。需要有明确的职权来做决定的例子包括：实施方法的选择，质量验收，以及如何应对项目偏差。在项目团队成员的职权水平与其职责水平一致时，其工作最富有成效。

（3）职责：指为完成项目，要求项目团队成员实施的工作。

（4）能力：指完成项目活动所需的技能和能力。如果项目团队成员不具备所需要的技能，绩效将受到影响。如果发现了这种不匹配的现象，则应采取提前的应对措施，例如，培训、招募、进度计划变更或范围变更。

3. 项目管理者

要开发一个项目，就要组建一个团队来实施，而作为团队的核心，首先是要确定这个项目的负责人（或称为项目经理）。

项目管理是集中于人的活动，项目负责人是项目组织的核心和项目团队的灵魂，对项目进行全面的管理。他的管理能力经验水平、知识结构、个人魅力都对项目的成败起着关键的作用。但是，能胜任开发的人员常常有可能是拙劣的项目负责人，他们完全没有管理人的技能。如Edgemon所说："很不幸而且是很经常地，似乎人们碰巧落在项目管理者的位置上，也就意外地成为了项目管理者"①。实际上，这种"天上掉馅饼"式的选择项目管理者，会给项目管理带来很多不确定性。

美国著名的企业家卡耐基说过："一个人的成功，只有15%靠他的专业技术，85%则靠他的人际处理能力。"但他同时又说，"软与硬是相对而言的。专业的技术是硬本领，善于处理人际关系的交际本领则是软本领。"这些话对于一般岗位上的管理者似乎是适用的。如果是经营一个加工厂或一个饭店，管理者们可以不必懂机器如何操作，也不必掌握高超的烹调

① Pressman. 软件工程——实践者的研究方法. 第6版. 北京：机械工业出版社

技艺。因为他们的常识，以及通过耳闻目睹或者咨询都能解决实践中的问题。但在软件领域中，超然于技术之上的管理者是无法生存的，不管他的职务是项目经理、程序经理、开发组长，还是部门经理，即使再具备管理能力，但无法保证自己在技术浪潮中安然无恙。软件公司的各级经理最好既精通技术又懂管理。作为技术型项目负责人，应理解项目中的各项技术细节，对团队中每个成员在做什么，做得怎么样，会遇到什么难题，这些难题可能有哪些解决方案，下一步要做什么，什么时候项目能做到什么样等有很好的把握。这样的项目负责人显然对项目的进度有很强的控制力，能有效地推进项目的进展，最终保证项目的成功。

当然，除应具备相应的技术背景外，项目管理者应具备与之相适应的管理能力，但这一点与技术、方法、过程相比，常被人们忽视。Bob Glass 更直言不讳地说："我一直认为管理是很烦人的话题。在我读过的有关管理的书籍中，95%都是常识，其余 5%是重温陈词滥调。"但他也不得不承认："好的管理比好的技术更重要。"软件项目管理者是工作的组织者，他的管理能力的强弱是项目成败的关键因素之一。寻找一个项目管理者并不容易，要找到一个适合项目的项目经理更加困难。

微软公司在选择经理人员时，总是把他们的技术知识和运用技术去赚钱的能力放在首位。挑选"顶尖的聪明人"做项目管理者是微软公司的择人标准。比尔·盖茨认为："聪明就是能迅速地、有创建地理解并深入研究复杂的问题。具体地说，就是善于接受新鲜事务，反应敏捷；能迅速地进入一个新的领域，并对其做出创建性的解释；提出的问题往往刺中要害；能及时掌握所学知识，并博学强识；能把原来认为互不相干的领域联系在一起并使问题得到解决。"看来，比尔·盖茨给了我们一个用人标准，项目管理者就是团队中最聪明的那个。一个有活力的软件公司的各级管理者都不会有这样的感叹，"因为我啥也不会干，所以只好当管理者。"

3.2.4　产品

产品是指向市场提供的能满足人们某种需要的一切东西，包括实物、服务保证、创意、思想等各种有形或无形的形态，而软件产品属于后者。

1. 软件产品的范围

软件计划的第一个任务就是确定软件的工作范围，即软件的用途及对软件的要求。在这里，范围的概念包含两方面，一个是产品范围，即产品或服务所包含的特征或功能，另一个是项目范围，即为交付具有规定特征和功能的产品或服务所必须完成的工作。

在确定范围时首先要确定最终产生的是什么，它具有哪些可清晰界定的特性。要注意的是特性必须要清晰，以认可的形式表达出来，如文字、图表或某种标准，能被项目参与者理解，绝不能含含糊糊、模棱两可，在此基础之上才能进一步明确需要做什么工作才能产生所需要的产品，也就是说产品范围决定项目范围。

范围说明在项目参与者之间确认或建立了一个项目范围的共识，作为未来项目决策的文档基准。范围说明中至少要说明项目论证、项目产品、项目可交付成果和项目目标。

软件范围是通过回答下列问题来定义的。

背景：待建造的软件如何适应于大型的系统、产品或商业的背景，在该背景下要加什么约束？

信息目标：软件要产生什么样的客户可见的数据对象来作为输出使用？需要什么样的数据对象作为输入？

功能和性能：对于软件功能的要求，在某些情况下要进行求精细化，以便能够提供更多的细节，因为成本和进度的估算都与功能有关。软件的性能包括处理时间的约束、存储限制以及依赖于机器的某些特性。要同时考虑功能和性能，才能做出正确的估计。软件要执行什么样的功能使输入数据能变换成为输出数据？是否需要满足什么特殊的性能特征？

计划人员必须使用管理人员和技术人员都能理解的无二义性的语言来描述工作范围。对软件范围的描述必须是界定的，即明确给出定量的数据（如并发用户数目、邮件列表的大小、允许的最大响应时间），说明约束和/或限制（如产品成本、内存大小），描述其他的特殊因素。

2. 问题分解

问题分解，有时称为划分，其认识基础来源于西方哲学的还原论。这种思想可以追溯到古希腊哲学家德谟克利特①，其思想本质是在不断线性的追问中把世界的本质还原为一个原子本因，并在这一还原过程中建立一种绝对的因果性来解释世间万物。兴起于西方世界的整个近代科学，几乎均为依赖于还原论方法的辉煌成就。运用还原论方法研究自然，再把获得的知识纳入公理化演绎体系加以表达，已成为科学研究的标准模式。或者说，科学方法本质上就是还原论方法。

著名物理学家史蒂芬·霍金在他所著的时间简史中描述了问题划分的思想："毕全功于一役设计一种能描述整个问题的理论，看来是非常困难的。反之，我们是将问题分成许多小块，并发明许多部分理论，每一部分理论描述和预言一定有限范围的观测，同时忽略其他量的效应或用简单的一组数表示之。"②

大多数问题是庞大的，有时处理起来非常棘手，特别是如果它们提出了某些以前从没解决过的新东西。面对复杂的问题人类常常采用分而治之的策略。简单地讲，就是将一个复杂的问题划分成若干能够理解并较易处理的小问题，这是项目计划开始时所采用的策略。为了分析问题，我们用问题集和它们间的相互关系来描述。图 3-4 解释了如何划分问题的过程。重要的是要记住关系（图中的箭头，及子问题的相对位置）跟子问题本身一样重要。有时，正是关系保持着怎样解决大问题的线索，而不简单是子问题的本身特性。

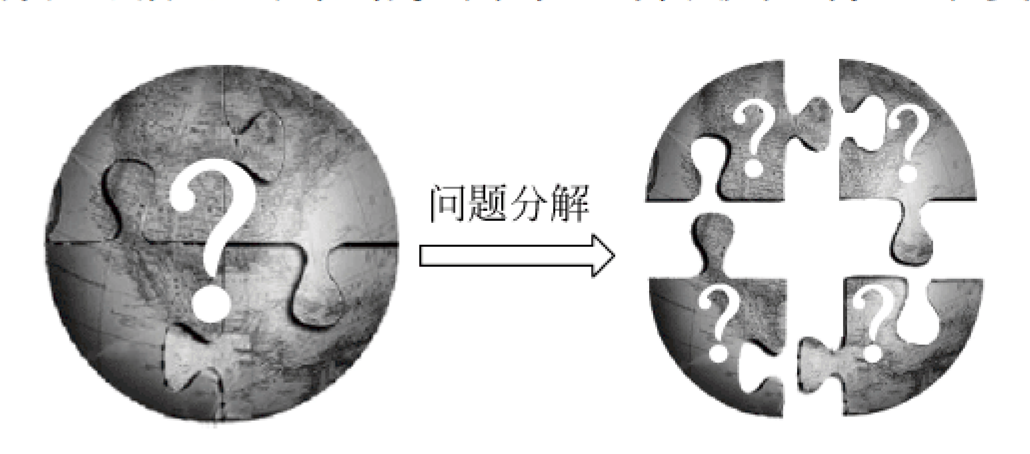


图 3-4 问题的分解方法

在确定软件范围的活动中并没有完全分解问题。分解一般用于两个主要领域：①必须交付的功能；②交付所用的过程。在估算开始之前，范围中所描述的软件功能必须被评估和精化，以提供更多的细节。因为成本和进度估算都是面向功能的，所以某种程度的分解是

① 德谟克利特（希腊文 Δημόκριτος 英文 Demokritos，约公元前 460—前 370），古希腊伟大的唯物主义哲学家，原子唯物论学说的创始人之一。

② 史蒂芬.霍金.时间简史.许明贤，译.湖南：湖南科学技术出版社，1994：21

很有用的。

但是,任何一个系统的属性是不能被部分属性所决定的,系统的整体一定大于部分之和。最终问题只能通过综合才能解决,这是系统观的整体论思想。例如,当软件功能被分解成许多子模块后,必须将其集成起来,才能形成系统的整体功能。这就是整体论与还原论相结合的思想。

3.2.5 过程

软件开发项目的整个过程充满了不确定性。软件开发是一个知识创造的过程,每次软件开发项目的目标和手段都不同,没有完全相同的模式可供复制。其创造性过程的本质决定了软件开发项目包含了许多不确定的方面和未知的因素。

在第2章中已经介绍了许多过程模型,一旦选定了过程模型,公共过程框架(Common Process Framework,CPF)就应该适于它。CPF可以用于线性模型,还可以用于迭代和增量模型、演化模型,甚至是并发或构件组装模型。CPF是不变的,充当一个软件组织所执行的所有软件工作的基础。

软件过程提供了一个框架,在该框架下可以建立一个软件开发的综合计划。若干框架活动适用于所有软件项目,而不在乎其规模和复杂性。若干不同的任务集合——每一个集合都由任务、里程碑、交付物以及质量保证点组成——使框架活动适应不同软件项目的特征和项目组的需求。最后是保护性活动——如软件质量保证,软件配置管理和测度——它们贯穿于整个过程模型之中。保护性活动独立于任何一个框架活动,且贯穿于整个过程之中。

3.2.6 项目

项目是一个特殊的将被完成的有限任务,它是在一定时间内,满足一系列特定目标的多项工作的总称①。

1. 项目的特性

软件开发项目是由特定的组织体来进行的,一般来说具有下列特性。

(1) 项目具有时限性和独特性。一般性日常工作是连续性和重复性的,而项目则是有时限性和唯一性的。软件开发项目是有限时间内的组织体所进行的开发活动,是通过项目经理及其团队合作完成的工作。每个项目都是独特的,或者其提供的成果有自身的特点,因此项目总是独一无二的。

(2) 项目以明确的目标为导向。软件开发项目以适应应该解决的问题或题目而组织起来,并且明确该项目应该达到的目标,项目的成功与否决定于该目标是否已达成。

(3) 项目存在大量的变更管理。软件项目开发过程中往往伴随着许多不确定性的因素,为了克服这些不确定性,需要采取必要的随机应变的体制,以适应软件开发项目的状况。

(4) 活动的整体性/过程的渐进性。项目中的一切活动都是相互联系的,构成一个整

① 白思俊.现代项目管理.北京:机械工业出版社,2002

体，不应有多余的活动，也不能缺少某些活动，否则必将损害项目目标的实现。项目活动是一个整体，但各阶段之间是依次渐进、有时可能是循环递进发展的。项目的实施同样需要逐步地投入资源，持续地累积可交付成果，始终要精工细作直至项目的完成。

2. 项目管理三角形

抛开软件项目的特殊性，就一般项目管理而言，项目管理三角形(Project Management Triangle)的理论都是普遍适用的。它是指项目管理中范围、时间、成本 3 个因素之间的互相影响的关系，而项目的质量是由这 3 个因素的平衡关系所决定的，如图 3-5 所示。

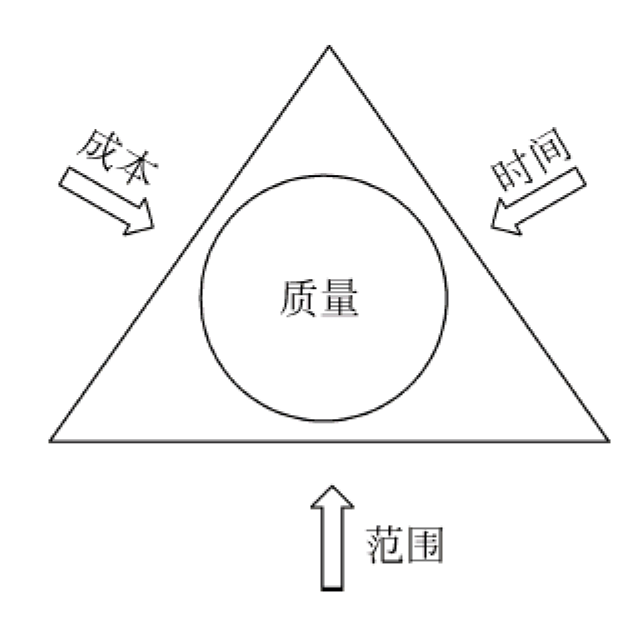


图 3-5 项目管理三角形理论

(1) 为了缩短项目时间，就需要增加项目成本(资源)或减少项目范围；但是，当调整项目时间时，项目成本可能增加，资源可能会被过度分配，而且项目范围也可能发生变化。

(2) 为了节约项目成本(资源)，可以减少项目范围或延长项目时间；如果需求变化导致增加项目范围，就需要增加项目成本(资源)或延长项目时间。

(3) 如果调整项目三角形的范围边，改变项目的范围一定包括改变项目任务的数量和工期。项目范围和质量是密切相关的，在缩小范围的同时，会降低既定的项目质量要求，否则不可能在原来的资源和时间内达成新的目标，所以项目的预期目标限定了相应的资源和时间。

质量处于项目三角形中的中心位置，项目三角形的 3 条边中任何 1 条边发生变化都会影响项目质量。例如，如果发现项目工期还有剩余时间，可以通过增加项目任务来扩大范围。有了这种项目范围的扩大，就能够提高项目质量。反之，如果需要降低项目成本，将其控制在项目成本范围之内，就不得不通过减少项目任务或者缩短项目工期来缩小项目范围。随着项目的缩小，就很难保证既定的项目质量，所以削减项目成本会导致项目质量的降低。

项目作为一个整体，要使各方面的资源能够协调一致，就要特别熟悉项目三角形的概念。项目三角形中的范围，除了要考虑对项目直接成果的要求，还要考虑与之相关的在人力资源管理、质量管理、沟通管理、风险管理等方面的工作要求。所以，项目管理三角形涉及 PMI 项目管理知识体系中的全部知识领域①。

3.3 软件项目管理的活动——从这里开始

项目管理是一系列的伴随着项目的进行而进行的，目的是为了确保项目能够达到期望的结果的一系列管理行为。而软件项目管理是为了使软件项目能够按照预定的成本、进度、质量顺利完成而进行分析和管理的活动。

那么围绕项目三角形要开展哪些管理活动呢？软件项目管理涉及的活动很多，也很繁杂。每一项活动的内容都足以用一本书来专门叙述。这里，我们参考美国项目管理学会

① 项目管理知识体系是美国项目管理研究所(PMI)开发的一个关于项目管理的标准，已经成为美国项目管理的国家标准之一，也是当今项目管理知识与实践领域的事实上的世界标准。

(PMI)提出的项目管理框架①,结合软件工程的实践,分别叙述软件项目管理的主要内容。

3.3.1　软件项目管理的活动概述

软件项目管理的对象是软件工程项目,因此软件项目管理涉及的范围将覆盖整个软件工程过程。软件项目管理包含的主要活动有:项目沟通(需求获取)、软件项目计划、项目范围、项目估算、进度管理、软件质量保证、软件过程能力评估、软件配置管理、风险管理等。这些活动都是贯穿、交织于整个软件开发过程中的。

在软件项目管理过程中一个关键的活动是制定项目计划,它是软件开发工作的第一步。项目计划的目标是为项目负责人提供一个框架,使之能合理地估算软件项目开发所需的资源、经费和开发进度,并控制软件项目开发过程按此计划进行。

项目范围是指项目要获得什么产品或服务,项目要努力实现的目标,生产项目产品或服务所能包括的所有工作和生产这些产品或服务所用的过程。项目的范围管理则是保证项目团队与项目利益相关者(包括项目当事人和其利益受该项目影响(受益或受损)的个人及组织)对项目所生产的产品或服务有一致共同的理解,对项目生产什么和不生产什么进行有效的定义和控制。

在做计划时,必须就需要的项目成本、项目进度(项目持续时间)做出项目估算。软件项目估算从来都没有成为一门精确的科学,因为变化和不确定性的东西太多。但是,软件项目的估算还是能够通过一系列系统化的步骤,在可接受的风险范围内提供估算结果。

软件过程能力评估是对软件开发能力的高低进行衡量;软件配置管理针对开发过程中人员、工具的配置、使用提出管理策略。软件过程能力描述了一个开发组织开发高质量软件产品的能力。现行的国际标准主要有两个:ISO 9000.3 和 CMM。ISO 9000.3 是 ISO 9000 质量体系认证中关于计算机软件质量管理和质量保证标准部分。

软件质量保证(Software Quality Insurance,SQA)是在软件过程中的每一步都进行的“保护性活动”,正式的技术评审是最为重要的 SQA 活动之一。

软件配置管理(Software Configuration Management,SCM)是应用于整个软件过程中的保护性活动,它是在软件整个生命周期内管理变化的一组活动。软件配置由一组相互关联的对象组成,这些对象也称为软件配置项,它们是作为某些软件工程活动的结果而产生的。一旦一个配置对象已被开发出来并且通过了评审,它就变成了基线。对基线对象的修改将导致该对象新的版本建立。版本控制是用于管理这些对象而使用的一组规程和工具。

项目风险管理是指对项目风险从识别到分析乃至采取应对措施等的一系列过程,是项目管理中很重要的管理活动。风险管理的过程应贯穿整个项目生命周期,有效地实施软件风险管理是软件项目开发工作顺利完成的保证。

以上介绍的与软件项目管理相关的内容将在本章及后面的章节中加以论述。

3.3.2　项目沟通与需求管理

项目沟通管理是现代项目管理知识体系中的九大知识领域之一。沟通是一个过程,在这个过程中,信息通过一定的符号、标志或者行为,在个人之间、团队之间、组织之间进行交

① 项目管理知识体系指南.第 3 版

换。项目沟通管理为成功所必需的因素——人、想法和信息之间提供了一个关键连接。涉及项目的任何人都应以准确的项目“语言”发送和接收信息，不准确的信息不仅毫无价值，而且可能引起误解，造成混乱。项目管理中，沟通是一个软指标，沟通所起的作用不好量化，沟通对项目的影响往往也是隐形的。然而，沟通对项目的成功，尤其是软件工程项目成功的重要性却不可低估。

对于项目来说，要科学地组织、指挥、协调和控制项目的实施过程，就必须进行信息沟通。沟通对项目的影响往往是潜移默化的，所以，在成功的项目中人们往往感受不到沟通所起的重要作用，在失败项目的痛苦反思中，却最能看出沟通不畅的危害。没有良好的信息沟通，对项目的发展和人际关系的改善，都会存在着制约作用。

沟通失败是软件项目求生路上最大的拦路虎。常能听到的典型案例是某某企业耗资上千万元的项目(如 ERP 项目：企业资源规划)最终弃之不用，原因是开发出的软件不是用户所需要的，没提高用户的工作效率反而增加了工作量。不难看出，造成这种尴尬局面的根本原因是沟通失败。当一个项目组付出极大的努力，而所做的工作却得不到客户的认可时，是否应该冷静地反思双方之间的沟通问题？软件项目开发中最普遍的现象就是一遍一遍地返工，导致项目的成本一再加大，工期一再拖延，为什么不能一次把事情做好？原因还是沟通不到位。

人们通常持有一个错误概念，在项目沟通时(需求分析阶段)，开发者必须完全理解与确定客户想要什么样的软件，然而大多数情况下这个目标很难实现。当客户向需求分析人员提出需求时往往是通过自己的想法用自然语言来表达的，这样的表达结果对于真实的需求来说是一种描述(甚至只是某个角度的描述)，远远不能保证这样的描述能够准确地传达给开发者，也许在同客户交流的第一时刻就埋下了理解分歧的种子。以上事实让我们联想起一个有趣的事件：历史上(1967 年)一位美国总统候选人不止一次在言辞上犯错出丑，他还为此专门召开了一个新闻发布会并宣布：“我知道你相信你理解了你认为我所说的，但是我不能确定你是否认识到你听到的并不是我所意指的。”

尽管这句话听起来有点绕，但这个托词很适用沟通的问题。让我们再看一幅在网络上流传甚广的关于项目沟通的图片(图 3-6)，它形象地说明了一个需求不明确的项目是如何可笑地演变成最终产品的。

这个图示故事告诉我们，开发人员听到了客户要求，但他们所听到的不一定是客户想说的。任何管理者如果在项目开发早期没有真实有效地理解用户意图，他有可能为错误的问题提供并建造一个花费巨大而不切实际的解决方案。对过程不在意的管理可能造成把有效的技术方法和工具送入到“真空”中的风险。

软件需求是整个软件项目的最关键的一个输入，和传统的生产企业相比较，软件的需求具有模糊性、不确定性、变化性和主观性的特点，它不像生产汽车、计算机等硬件的需求，是有形的、客观的、可描述的、可检测的。软件需求是软件项目中最难把握的问题，同时又是关系项目成败的关键因素，因此对于需求分析和需求变更的处理十分重要。

面对软件工程中的需求不确定性，软件工程进一步发展，提出了“需求工程”的概念。需求工程提供了一种适当的机制，以了解用户需要什么，分析需求、评估可行性、协商合理的方案，并无歧义地详细说明方案。现代需求工程一般被描述为以下 6 个步骤。

(1) 获取(需求诱导)。

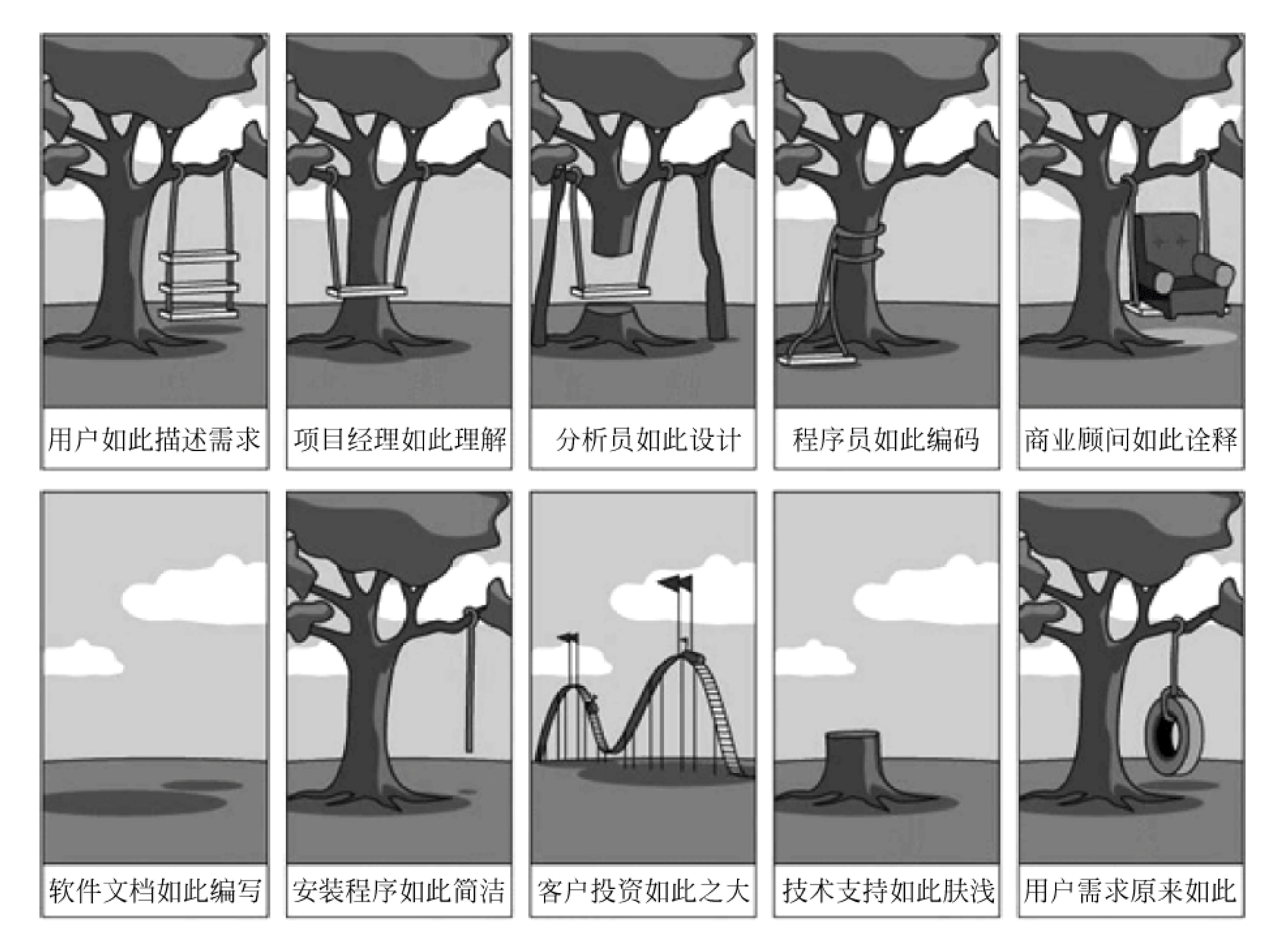


图 3-6 项目沟通的故事

(2) 分析(需求分析和商谈)。

(3) 规定(规约)。

(4) 系统建模。

(5) 验证(需求确认)。

(6) 需求控制(控制与变更管理)。

需求管理要形成的这一阶段的成果是需求规格说明书、功能测试方案和验收评审标准。需求规格说明书包括业务模型、用例分析、各种流程等。有关需求分析与工程化的内容,在本书第 2 篇和第 3 篇中将进行介绍。

3.3.3 软件项目计划的制定

每一个项目启动时,都是一个新的挑战。一个良好的计划对项目的管理至关重要,但并不容易。有人说,做项目计划,如同给一个待出生的婴儿写传记那样困难。此话虽有些言过其实,但也比喻形象。

项目计划定义了工作的内容以及如何完成这些工作。它为每一个主要任务下了定义,还对时间和所需资源进行了估算,也为管理评审和控制提供了框架[①]。制定计划的意义在于可以真实客观地反映项目全貌,及时地发现问题,纠正问题,确保项目可严格按照计划执行。项目的进展是由众多因素来构成的,所以,如何快速地了解项目的真实情况,并加以控

---

① Watts S. Humphrey. 软件工程规范. 傅为,苏俊,许青松,译. 北京:清华大学出版社,2003

制，制定计划是一个最好的手段。

制定软件项目计划没有固定的格式以供参考，它更多是一种经验性的技巧。许多项目管理手册提供了一个模板式的框架，可作为用户参考。一个良好的软件项目计划应包括的基本内容有：项目范围、活动的分解、项目的组织结构、项目进度、项目成本估算、软件质量保证计划、配置管理计划、资源管理计划、测试计划、培训计划、安全计划、风险管理计划以及维护计划。

制定了一个良好的项目计划，并不意味着就可以成功地完成项目，前面介绍了项目是由众多因素构成的，制定了项目计划仅仅是项目的开始。所以，项目应该有个好的开始，并努力朝着好的方向发展。

项目计划的制定过程是一个多次反复的动态调整过程。在项目执行过程中，当项目的某一个因素发生变更时，往往会直接影响到其他因素，需要同时考虑一项变更给其他因素造成的影响，项目的控制过程就是要保证项目各方面的因素从整体上能够相互协调。

3.3.4 项目范围与管理

由项目的定义可知，任何项目都必定有一个范围。项目范围管理对于项目的成功来讲是十分关键的，确定不了范围，项目就无法启动，也无法按项目进行管理，计划、进度、工期就无从谈起，成本管理、资源保障、变更控制等就失去了根据。

项目范围也称为工作范围，即为使客户满意而必须做的所有工作，它包括项目的最终产品或服务以及实现该产品或服务所需要做的各项具体工作。在实际项目中，产品范围和项目范围的含义是不同的，产品范围是确定产品或服务中应包含有哪些功能和特征，是对产品要求的度量；项目范围是为了使客户满意而必须做的所有工作，是项目未来一系列决策（项目计划）的基础。

范围管理就是为了成功地实现项目的目标，规定或控制哪些方面是项目应该做的，哪些是不该做的，即定义项目的范畴。简单地说，就是项目要做什么、怎么做、做到什么程度。“做什么”通常是由项目客户（项目使用者、投资人、行政主体等）给出的，一般指明了产品的范围；“怎么做”是由项目团队为实现项目目标而做出的，是项目的工作范围；“做到什么程度”是项目团队根据项目范围给定的条件，为实现某一类工作或某个“工作包”而做的具体工作①。

要想真正管理好项目范围管理，没有必要的技术和好的方法是肯定不行的。项目管理知识体系指南指出，有效的项目范围管理，一般需进行以下过程：范围计划，范围定义、工作分解结构、范围核实、范围控制。

首先强调的就是要周密地做好范围计划编制。范围计划编制是将产生项目产品所需进行的项目工作（项目范围）渐进明细和归档的过程。这些文档中包括用于衡量一个项目或项目阶段是否已经顺利完成的标准等。作为范围计划过程的输出，项目组要制定一个范围说明书和范围管理计划。做项目范围计划编制工作是需要参考很多信息的，如产品描述，首先要清楚最终产品的定义才能规划要做的工作。项目沟通活动通常已对项目范围有了粗线条

---

① 李卫星．突破项目管理的难点：从 WBS 到计划．北京：电子工业出版社，2006

的约定，范围计划应在此基础上进一步深入和细化。

良好的范围定义能够明确项目相关利益者（用户、项目管理人员与开发人员、其他相关人员）的利益与责任，保证项目的正常开展，对项目的进度、费用、质量等都将产生积极的影响。定义项目范围之后，为了便于管理和可操作，还必须将项目或子项目进行逐层分解，直至分解成可以操作和管理的子任务，每个子任务相对易于完成并便于管理。

范围核实是指项目的范围需要得到项目相关人的一致认可和接受，并形成书面文件。现实软件开发过程中常常出现这种现象：开发人员费尽心力开发出来的软件产品最后得不到用户的认可或完全认可，其中一个重要的原因就是未进行范围核实。

世界万物都处在不断的变化之中，只有变化是永恒的。项目范围也会因为环境、用户需求、市场、管理、技术等诸多因素而变更。无论怎样的变更，都会对项目产生影响，并且对项目开发者来讲，影响常常是负面的。如何控制项目范围变更，使项目变更尽量朝着有利于项目进展的方向发展，这就是项目变更控制的内容。范围控制的输出成果主要是更新的项目范围说明书、工作分解结构词汇表、批准的变更请求、推荐的措施经验即教训、项目管理计划。这些成果都应该记录到变更控制系统或配置管理系统，作为项目管理的依据。

3.3.5　工作分解结构

亚里士多德认为：人们可以将物质无限制地分割成越来越小的小块，即人们永远不可能得到一个不可再分割的最小颗粒。这也许是最早的系统还原论的思想。

工作分解结构（Work Breakdown Structure，WBS）的建立对项目范围管理来说意义非常重大，它使原来看起来非常笼统、非常模糊的项目目标一下子清晰起来，使项目管理有依据，项目团队的工作目标清楚明了。

1. WBS 的意义

要让项目计划贴近现实，按照系统还原论的思想，首先需要把项目中所有的工作都罗列出来，然后把每一个步骤的工作进行细分，一直细分到“原子级”（图 3-7）。在项目规划的过程中，人们往往会求助于 WBS 方法进行项目工作内容的分解。在此基础之上再进行资源的分配、进度计划并估计项目的成本。

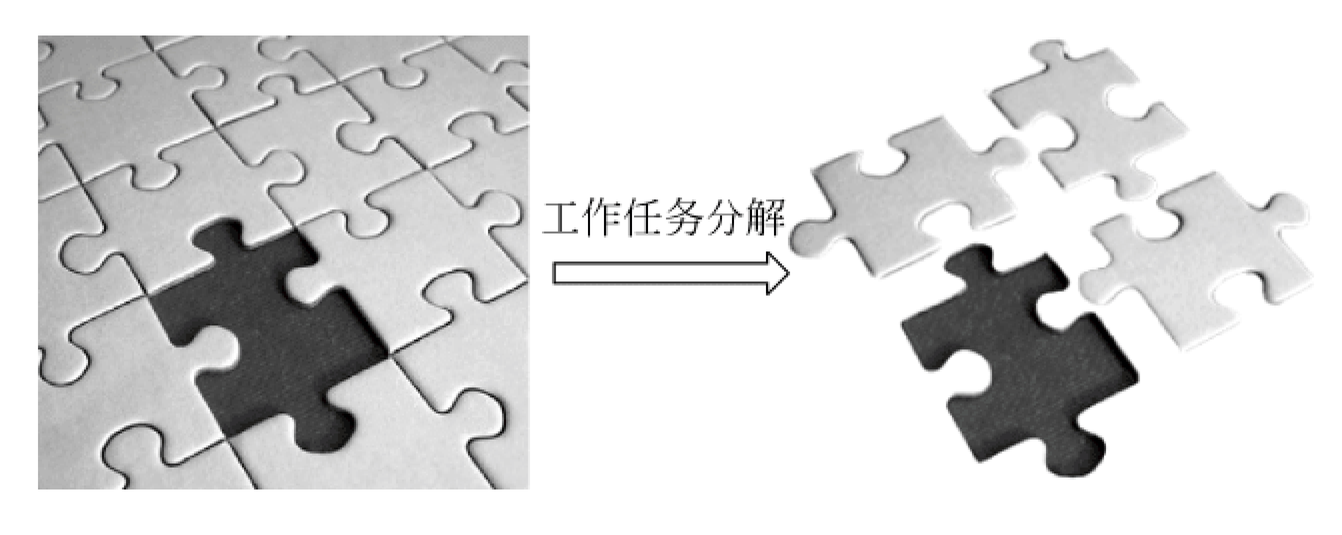


图 3-7　工作任务分解

WBS 是一个详尽的、层次的（从全面到细节）树形结构，由明确的可提交项目目标与为了完成项目需要执行的任务组成。WBS 的目的是识别项目中实际需要完成的工作。WBS 把项目工作分成较小，更便于管理的多项工作，每下降一个层次便是对项目工作进行更详尽

的说明。WBS通常是一种面向"成果"的"树",其最底层是细化后的"可交付成果"。但WBS的形式并不限于"树"状,还有多种形式。

在核实工作分解程度必要而充分后,可将这些被细化的处于最底层的分项工作安排在进度计划中,在可预估成本的前提下,对其质量进行监督和控制。从而掌握对整个项目目标在宏观进度,成本和质量上的控制权。因此,WBS是项目计划的基础。

在项目管理实践中,WBS处于计划过程的中心,也是制定进度计划、资源需求、成本预算、风险管理计划和采购计划等的重要基础。WBS同时也是控制项目变更的重要基础。

2. 创建 WBS

创建WBS是指将复杂的项目分解为一系列明确定义的项目工作并作为随后计划活动的指导文档。创建WBS的过程非常重要,因为在项目分解过程中,项目经理、项目成员和所有参与项目的相关利益者都必须考虑该项目的所有方面。

制定WBS的过程如下。

(1) 得到范围说明书。

(2) 召集有关人员,集体讨论所有主要项目工作,确定项目工作分解的方式。

(3) 分解项目工作。如果有现成的模板,应该尽量利用。

(4) 画出WBS的层次结构图。WBS较高层次上的一些工作可以定义为子项目或子生命周期阶段。

(5) 将主要项目可交付成果细分为更小的、易于管理的工作包。工作包必须详细到可以对该工作包进行估算(成本和历时)、安排进度、做出预算、分配负责人员或组织单位。

(6) 验证上述分解的正确性。如果发现较低层次的项没有必要,则修改组成成分。

(7) 随着其他计划活动的进行,不断地对WBS更新或修正,直到覆盖所有工作。

WBS的分解可以采用多种方式进行,包括按产品的物理结构分解、按产品或项目的功能分解;按照实施过程分解及按照项目的各个目标分解。图3-8是WBS的一般性示意图。

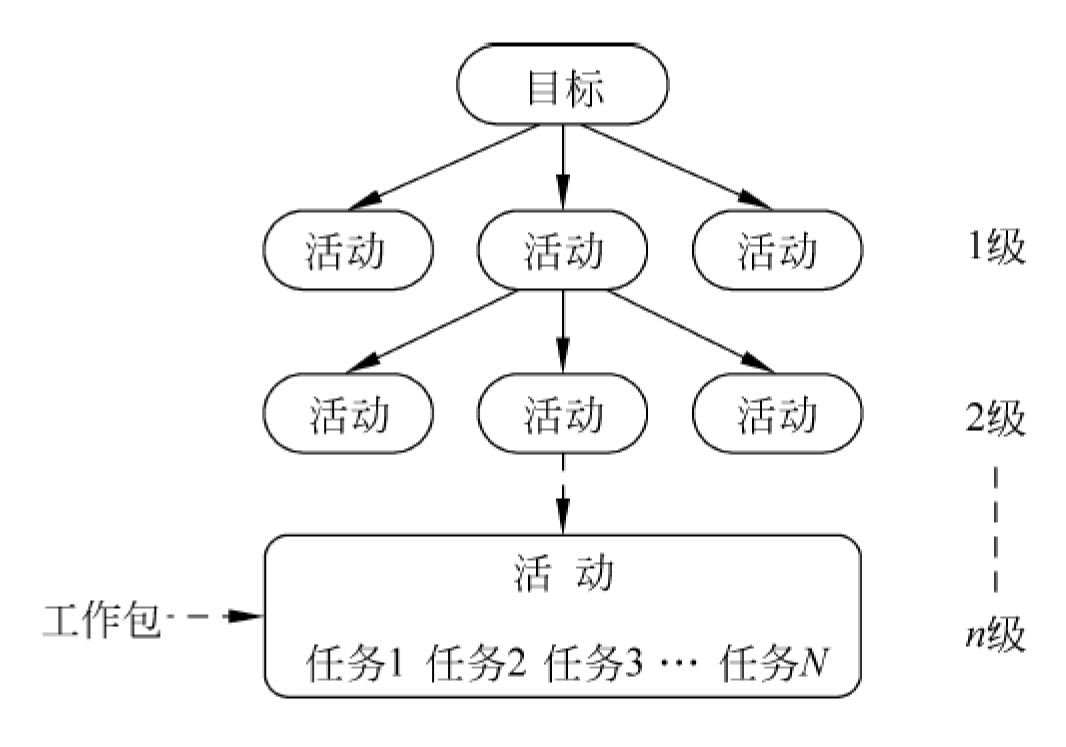


图 3-8 WBS的一般性示意图

创建WBS的方法一般有两种:一是自上而下的方法,从项目的目标开始,逐级分解项目工作,直到参与者满意地认为项目工作已经充分地得到定义。该方法由于可以将项目工作定义在适当的细节水平,对于项目工期、成本和资源需求的估计可以比较准确。二是采用

自下而上的方法,从详细的任务开始,将识别和认可的项目任务逐级归类到上一层次,直到达到项目的目标。这种方法存在的主要风险是可能不会完全地识别出所有任务或者识别出的任务过于粗略或过于琐碎。

WBS 的最低层次的项目可交付成果称为工作包。工作包可以分配给另一位项目经理进行计划和执行,通过子项目的方式进一步分解为子项目的 WBS。工作包可以在制定项目进度计划时,进一步分解为活动。

3.3.6　软件项目的组织

将参加软件项目的人员组织起来,发挥最大的工作效率,对成功地完成软件项目极为重要。开发组织采用什么形式,要针对软件项目的特点来决定,同时也与参与人员的素质有关。

Boehm 在软件工程的第 7 条基本原理中提出:开发小组的人员应该少而精。这条基本原理的含义是,软件开发小组的组成人员的素质应该好,而人数则不宜过多。开发小组人员的素质和数量是影响软件产品质量和开发效率的重要因素。素质高的人员的开发效率比素质低的人员的开发效率可能高几倍至几十倍,而且素质高的人员所开发的软件中的错误明显少于素质低的人员所开发的软件中的错误。

软件项目组织结构的模式通常有以下 3 种可供选择。

(1) 按课题划分的模式。把软件人员按课题组成小组,小组成员自始至终参加所承担课题的各项任务,负责完成软件产品的定义、设计、实现、测试、复查、文档编制,甚至包括维护在内的全过程。

(2) 按职能划分的模式。把参加开发项目的软件人员按任务的工作阶段划分成若干专业小组。待开发的软件产品在每个专业小组完成阶段加工(即工序) 以后,沿工序流水线向下传递。例如,分别建立计划组、需求分析组、设计组、实现组、系统测试组、质量保证组、维护组等。各种文档按工序在各组之间传递。虽然这种模式在小组之间的联系形成的接口较多,但便于软件人员熟悉小组的工作,进而变成这方面的专家。

(3) 矩阵形模式。这种模式实际上是以上两种模式的复合。一方面,按工作性质,成立一些专门部门,如开发部、业务部、测试部等;另一方面,每一个项目又有相应的经理人员负责管理。每个软件人员属于某一个专门组,又参加某一项目的工作。矩阵形模式如表 3-1 所示。

表 3-1　矩阵形模式

部门 项目	部门一 (例如业务部)	部门二 (例如开发部)	部门三 (例如测试部)
项目一	成员一	成员四	成员七
项目二	成员二	成员五	成员八
项目三	成员三	成员六	成员九

相对于传统职能结构而言,矩阵式组织结构则表现出很大的优势:矩阵式结构有利于资源在不同项目间灵活分配,组织能够适应不断变化的外界要求。例如,以前 IBM 是典型

的金字塔格局,单一按照区域、业务职能、客户、产品等元素来划分部门,对市场和客户的反应很慢。在 IBM 形成了立体多维矩阵后,加强了横向连接,充分整合资源,提高了反应速度。

矩阵式管理虽然有诸多好处,但是操作复杂是它最大的弱点之一。矩阵式管理的另一个问题就是沟通量大,需要有较强的"沟通管理"能力,否则就会掉入会议的漩涡中。

微软公司的"特性小组"模式

微软公司为每个基本的职能领域创立了工作思路,它使人们广泛地学习和分担责任,把人员组合成多职能小组来作为更大的项目小组的一部分进行工作。每次微软公司膨胀得太大时,盖茨就马上把它拆成小单位,每一单位的人数以 200 人为限。微软公司组织体系的一个明显优势是它给各小组提供了充分的自由,每个小组就是一个相对独立的开发中心,致力于把各类产品推向特定的市场。

在产品单位里,程序经理、开发员和测试员以"特性小组"形式并肩"作战"。典型的小组由一位程序经理(他通常从事不止一个"特性"的设计开发工作)和 3~8 位开发员(其中一人为组长)组成,同时还配备平行的测试小组,其成员与开发员组成"搭档"。用户培训专家则作为产品小组的成员进行工作。产品经理负责为产品单位内的营销部门和产品规划组选定人员。客户支持人员虽是另一个独立部门的成员,他们中的产品专家却与单个产品单位密切合作。而某些职能领域(如程序管理)依然难以准确定义,其他一些职能领域(如程序管理与开发、程序管理与产品管理、开发与测试)难以截然分开,微软公司也从未试图把它们分开。

这种小型化组织体系,任务责任明晰,分层管理的公司体系必然导致另一个问题的产生。那就是组织与组织间,管理层与管理层之间交流的困难,公司力量的分散。如果注意公司内部的交流与合作,就可以避免以上问题的存在。高效的互联网使我们的信息传递和处理更加便捷,这种交流方式可以将传统的层级管理扁平化,更加直接迅速地进行交流。基于网络的工作和通信方式可以发挥神奇的效用。

事实上,由于人的因素的重要性,以至于美国梅隆大学软件工程研究所专门开发了一个人员管理能力成熟度模型(PM-CMM),旨在"通过吸引、培养、鼓励和留住改善其软件开发能力所需的人才,增强软件组织承担日益复杂的应用程序开发的能力"。PM-CMM 与软件能力成熟度模型相配合,指导组织完成一个成熟的软件过程的创建。

3.4 项目进度管理

项目计划是走向目标的诺言,是实现成功的保证,但计划如果没有及时地跟进和协调,也就无异于空中楼阁。项目的主要特点之一即是有严格的时间期限要求。一个项目能否在预定的时间内完成,是项目管理最为重要的问题之一,安排进度计划的目的也正是为了控制时间和节约成本。

进度管理,可以从两个方面来理解,一方面是要制定一个可行而且高效率的计划,而另一方面则是要执行此计划。但如果计划制定得不可行,恐怕一切都是空谈。同时,计划如果制定得过于宽松,进度管理也就自然失去了其存在的意义,不会对项目的顺利进行产生积极作用。下面一则故事告诉了我们这些事实。

关于项目进度的故事①

有一位程序员正忙着编写程序,经理问他还需要多久才能完成。

“明天就可以完成。”程序员立即回答。

“我想这是不切实际的,实话实说,到底还要多少时间?”经理说。

“我还想加进一些新的功能,这需要花两个星期。”程序员想了一会儿说。

“即使这样也期望过高了,只要你编完程序时告诉我一声,我也就满足了。”经理说。

几年以后,经理要退休了。在他去退休午餐会时,发现那位程序员正趴在机器旁睡觉:可怜的家伙整个晚上都在忙于编写那个程序。

3.4.1　项目里程碑

管理者需要信息。由于软件产品是无形的,信息只能以文档的形式获得,这些文档能够描述正被开发的软件的状况。没有这些信息,就不可能对项目进展和成本估算做出判断,也不可能及时调控项目的进度安排。软件过程要分解成一系列相关的基本活动,在进行项目规划时,就应该建立一系列的项目里程碑。一个里程碑就是一项软件过程活动的起始或终结点,用来标记活动的进度。

在项目管理进度跟踪的过程中,给予里程碑事件足够的重视,往往可以起到事半功倍的效用,只要能保证里程碑事件按时完成,整个项目的进度也就有了保障。尤其是对于大型软件开发尤为重要。可交付的文档是交付给客户的项目成果,通常在项目的分析、设计等主要的项目阶段结束时,可交付的文档也是里程碑,但里程碑不需要交付。里程碑是项目内部的阶段性成果,可以供项目管理者检查项目的进展情况,里程碑不是向客户交付的东西②。

里程碑也给客户和项目组提供了一个重新确认项目范围的机会。在里程碑处对项目变更的建议、评估和确认工作可以及时响应客户提出的新需求并防范项目风险。

作为一个例子,图 3-9 给出了每个活动(项目里程碑)的主要输出,即可支付的文档。这里使用了原型开发方法来帮助验证需求,图中给出了每个活动(项目里程碑)的主要输出。这个项目的可交付文档是需求定义和需求描述。

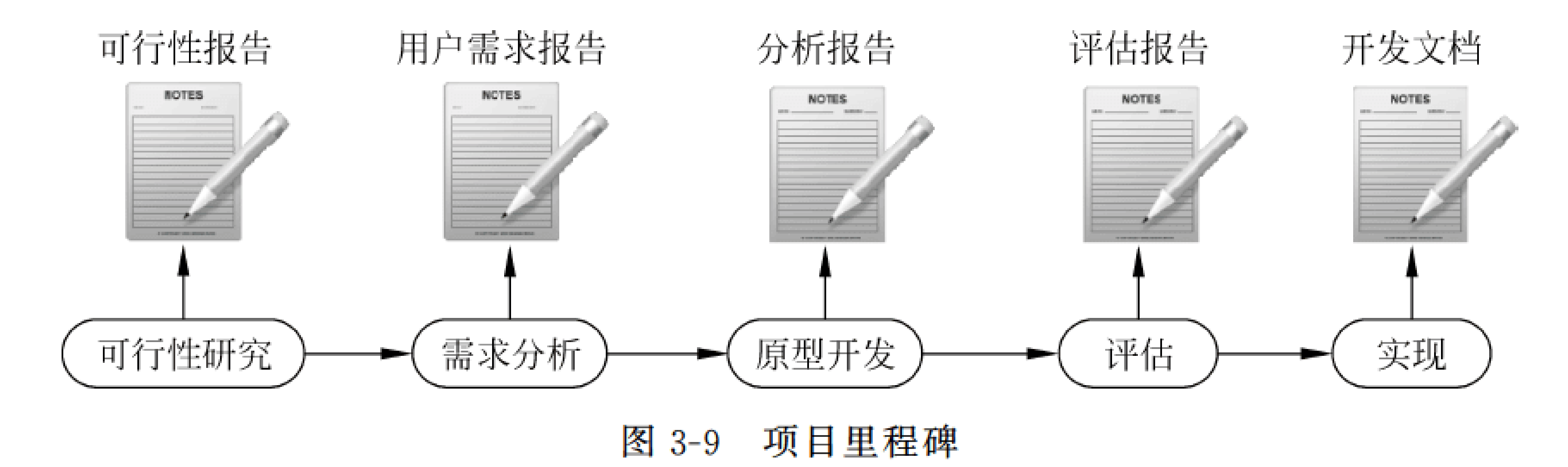


图 3-9　项目里程碑

微软公司的项目进度管理③

微软公司于 1988 年出品的 Publisher 1.0,是第一个在进度表中使用了里程碑的项目;微软公司将大项目分成若干里程碑式的重要阶段,各阶段之间有缓冲时间,但不进行单独的

① 林锐. 软件工程思想. CSDN 下载频道

② Sommerville. 软件工程. 程成,等译. 北京:机械工业出版社,2003

③ Michael A. Cusumano. 微软的秘密. 北京:北京大学出版社,1997:120

产品维护。1989 年和 1990 年的 Excel 3.0 是第一个采用此概念的大型项目。

微软公司的每一种软件产品都相应的有一个项目进度报表。高级行政管理人员(包括相关项目组的领导)每月都会接到从不同的项目组递交的此类报告。这些报告在使高级管理人员全面掌握各项目组的项目实施状况方面起到了关键性的作用。微软公司的总裁常常直接通过电子邮件与相关的经理人员或开发人员进行交流,并把所收集的信息用于正式的程序复查之中。项目组的成员同时也通过这种报告来设定自己下一步的目标。

微软公司的项目进度表由计划阶段与若干循环过程组成,每一循环包括产品的开发与发布,测试与稳定化,以及为意外事故准备的或在重要阶段连接处安插的“缓冲”时间,每个项目都有自己单独的进度表。但微软公司已经越来越强调产品之间发送安排的协调性。例如,Office 有一个集成的进度表,内容涵盖了 Excel、Word 以及其他作为套装应用软件一部分的产品。

Office 部门副总裁曾这样概述通常的进度:“一般说来,在总的进度表中,用一半的时间写出产品,留下另一半的时间调试或应付意外事故。这样,如果我有一个两年的项目,我会用一年来完成事先想好的东西……如果事情有点麻烦,我便去掉我认为不太重要的特性。”这种里程碑式的工作过程使微软公司的经理们可以清楚地了解产品开发过程进行到了哪一步,也使他们在开发阶段的后期有能力灵活地删去一些产品特性以满足发货时期的要求。

3.4.2 人员与工作量分配

1. 人员——进度权衡定律

著名学者 Putnam 在估算软件开发工作量时得出公式:$E=L^3(P^3t_d^4)$,其中 E 表示工作量,L 表示源代码行数,P 表示技术状态常数,t_d 表示开发时间。在这里,工作量的单位是人年,进度的单位是年。从公式中可知,软件开发项目的工作量(E)与交付时间(t_d)的 4 次方成正比,显然,软件开发过程中人员与时间的折中是十分重要的问题。Putnam 将这一结论称为“软件开发的权衡定律”。

2. Brooks 定律

曾担任 IBM 公司操作系统项目经理的 Brooks,从大量的软件开发实践中得出了另一条结论:“向一个已经拖延的项目追加开发人员,可能使它完成得更晚”。鉴于这一发现的重要性,许多文献称之为 Brooks 定律。它从另一个角度说明了“时间与人员不能线性互换”这一原则。

3. 用做人力计划的 Rayleigh-Norden 曲线

根据数以百计的大、小型软件开发项目的统计,对开发人员资源的需求(或称为消耗),包括对其他资源的需求,是随时间变化的一个类似于图 3-10 所示的曲线模式(以自然对数 e 为底的指数函数)。一开始资源需求量较小,然后逐渐上升,当到达某个时间常数(t_d)时需求量达到峰值,之后再逐渐下降,减少到较低的数值。因此,软件项目开发人员计划是一个时间的函数曲线,通常称为 Rayleigh-Norden 曲线。

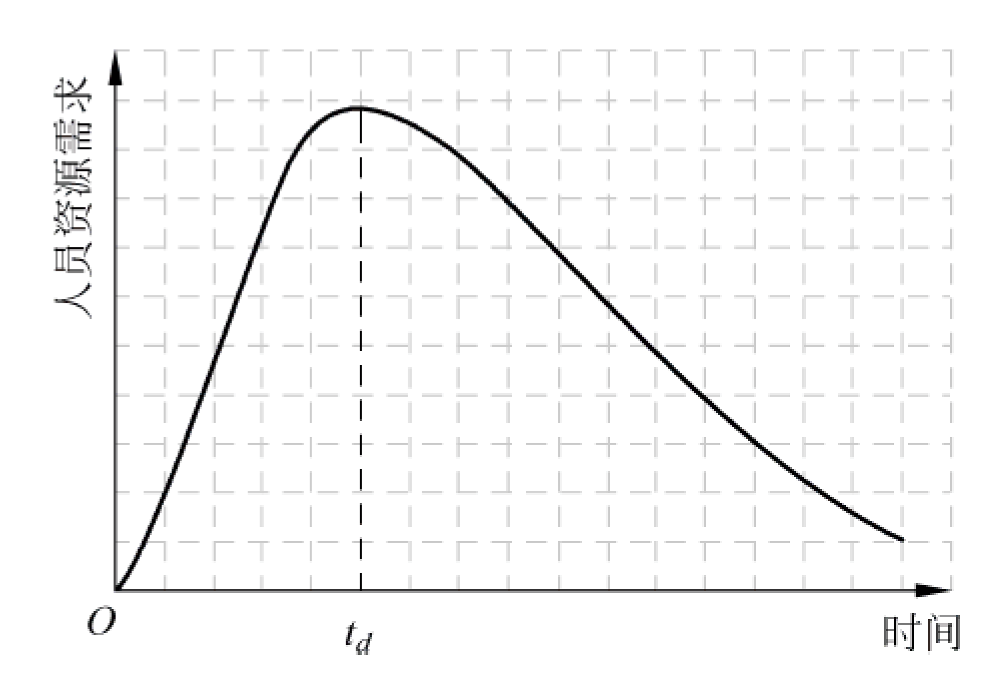


图 3-10 开发人员资源需求随时间变化的曲线

对上述 3 个定律的合理解读是：软件项目的建设时间主要取决于应用软件的开发时间，而人员与进度之间是一种非线性替代关系。当开发人员以算术级数增长时，人员之间的通信将以几何级数增长，而通信是要花费时间和成本的。当新的开发人员加入项目组之后，原有的开发人员必须向新来的成员详细讲解某个活动或工作包的来龙去脉。由于信息系统开发具有较强的个人风格，所以交流沟通的时间更容易拉长，而后来者还不一定能达到原来开发人员的工作质量，从而可能导致“得不偿失”的结果。

3.4.3　项目进度管理的可视化工具

项目进度通常用一系列的图表表示，通过这些图表可以了解任务分解、活动依赖关系和人员分配情况。

1. 甘特图

甘特图，也称为条状图，是在 1917 年由亨利・甘特开发的，其内在思想简单，基本是一条线条图，横轴表示时间，纵轴表示活动(项目)，线条表示在整个期间上计划和实际的活动完成情况。它直观地表明任务计划在什么时候进行，及实际进展与计划要求的对比。甘特图的优点是简单、明了、直观、易于编制，因此到目前为止仍然是小型项目中常用的工具。即使在大型工程项目中，它也是高级管理层了解全局、基层安排进度时有用的工具。

在如图 3-11 所示的甘特图中，可以看出各项活动的开始和结束时间。在绘制各项活动的起止时间时，也考虑它们的先后顺序。但各项活动之间的关系却没有表示出来，同时也没有指出影响项目寿命周期的关键所在。因此，对于复杂的项目来说，甘特图就显得难以适应。

ID	任务名称	开始时间	完成	持续时间	2009年10月										
					10	11	12	13	14	15	16	17	18	19	20
1	任务 1	2009-10-12	2009-10-20	7d											
2	任务 2	2009-10-12	2009-10-16	5d											
3	任务 3	2009-10-14	2009-10-16	3d											
4	任务 4	2009-10-12	2009-10-15	3d 4h											
5	任务 5	2009-10-19	2009-10-22	4d											

图 3-11　甘特图示例

2. 任务网络图

在安排项目进度时，可以根据 WBS 的分解情况，继续分解相应的活动(任务)，分析确定各个活动之间的顺序关系，画出任务的网络图。图中的每一项任务必须有一个前驱和后继，除了项目中的第一项和最后一项任务。开发者可遵循这些箭头，明确下一项工作任务，遇到里程碑，则知道此部分工作结束了。

一个项目的许多可视化特征都可以体现在任务网络图中。图 3-12 是一个任务网络图的示例，每一任务(活动)用有向连线将两个结点连接起来。连线上需标记任务代号和完成该任务所持续的时间长度(如周、天或月)，如 A：2 表示完成任务 A 需要的工作时间为 2 周(或 2 天)。

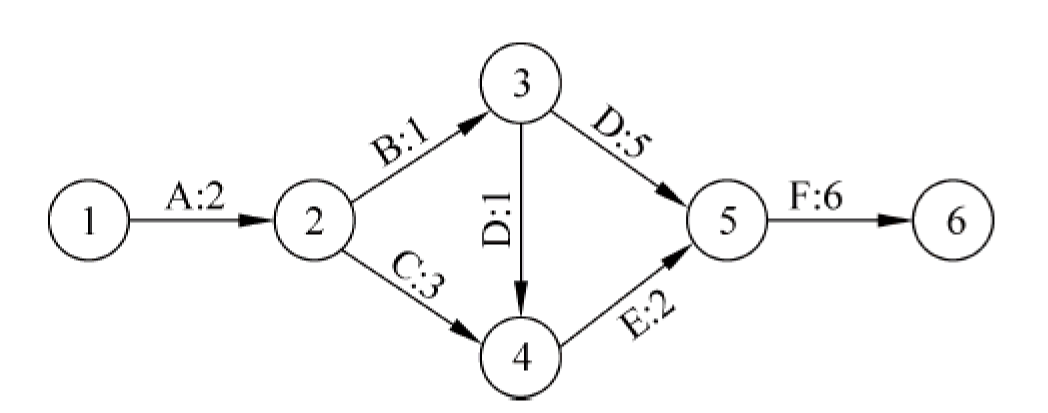


图 3-12 任务网络图的示例

需要认识到任务网络图是建立在对任务并行特征的理解基础上。任务网络图必须如实反映工作之间的并行性，如果所做的工作不能并行进行，那么任务网络图在描述工作之间的相互协调关系时所起的作用则不大。

任务网络图还可以确定关键路径在哪里，哪些任务有可能发生变化，然后结合资源、成本等情况，再不断进行资源调整优化以及工期、活动关系的调整等。

3.4.4 项目管理软件及其功能

目前市场上项目管理软件种类较多，具有代表性的是微软项目管理软件 Project 2000，其软件功能以美国项目管理协会(PMI)的项目管理理论为基础，包含了 PMBOK 九大知识领域中的五大核心领域，另外 4 个领域需要通过其他辅助工具或人工操作来完成。

项目管理软件一般提供如下功能①。

1. 预算及成本控制

大部分项目管理软件系统都可以用来获得项目中各项活动、资源的有关情况。人员的工资可以按小时、加班或一次性来计算，也可以具体明确到支付日期；对于原材料，可以确定一次性或持续成本；对各种材料，可以设立相应的会计和预算代码。另外，还可以利用用户自定义公式来运行成本函数。大部分软件程序都应用这一信息来帮助计算项目成本，在项目过程中跟踪费用。在项目过程中，随时可以就单个资源、团队资源或整个项目的实际成本与预算成本进行对比分析，在计划和汇报工作中都要用到这一信息。大多数软件程序可以随时显示并打印出每项任务、每种资源(人员、机器等)或整个项目的费用情况。

2. 进度安排

在实际工作中，由于项目规模往往比较大，人工进行进度安排活动就显得极为复杂了。项目管理软件包能为进度安排工作提供广泛的支持，而且一般是自动化的。大部分系统能根据任务和资源清单以及所有相关信息制作甘特图及任务网络图，对于这些清单的任何变化，进度安排会自动反映出来。

3. 图形

对于有大量活动事项的项目工程，人工制出一份甘特图或任务网络图，或人工进行修改

① IT 行业项目管理人士常用项目管理工具. http://pm.csai.cn(希赛网)

制图是一件极其乏味而又容易出错的工作。当前项目管理软件的一个最突出的特点是能在最新数据资料的基础上简便、迅速地制作各种图表,包括甘特图及网络任务图。有了基准计划后,任何修改就可以轻易地输入到系统中,图表自动会反映出这些改变。项目管理软件可以将甘特图中的任务连接起来,显示出工作流程。特别是用户可以仅用一个命令就在甘特图和网络任务图之间来回转换显示。

4. 导入/导出资料

许多项目管理软件包允许用户从其他应用程序,如文字处理、电子表格以及数据库程序中获得信息。为项目管理软件输入信息的过程叫做导入。同样地,常常也要把项目管理软件的一些信息输出到这些应用程序中,该过程叫做导出。

5. 处理多个项目及子项目

有些项目规模很大,需要分成较小的任务集合或子项目。另一种情况是经验丰富的项目经理同时管理好几个项目,而且,团队成员也同时为多个项目工作,在多个项目中分派工作时间。在这种情况下,大部分项目管理软件程序能提供帮助。它们通常可以将多个项目储存在不同文件里,这些文件相互连接。项目管理软件也能在同一个文件中储存多个项目,同时处理几百个甚至几千个项目,并绘制出甘特图和网络任务图。

6. 资源管理

目前的项目管理软件都有一份资源清单,列明各种资源的名称、资源可以利用时间的极限、资源标准及过时率、资源的收益方法和文本说明。每种资源都可以配以一个代码和一份成员个人的计划日程表。可以对每种资源加以约束,如它可被利用的时间、人员或成本。用户可以按百分比分为任务配置资源,设定资源配置的优先标准,为同一任务分配各个资源,并保持对每项资源的备注和说明。系统能突出显示并帮助修正不合理配置,调整和修匀资源配置。大部分软件包可以为项目处理数以千计的资源。

7. 计划

在所有项目管理软件包中,用户都能界定需要进行的活动。正如软件通常能维护资源清单一样,它也能维护一个活动或任务清单。用户对每项任务选取一个标题、起始与结束日期、总结评价,以及预计工期,明确与其他任务的先后顺序关系以及负责人。通常,项目管理软件中的项目会有成百上千个相关任务。另外,大部分程序可以创建工作分解结构,协助进行计划工作。

8. 项目监督及跟踪

项目管理的一项基本工作是对工作进程、实际费用和实际资源耗用进行跟踪管理。大部分项目管理软件包允许用户确定一个基准计划,并就实际进程及成本与基准计划里的相应部分进行比较。大部分系统能跟踪许多活动,如进行中或已完成的任务、相关的费用、所用的时间、起止日期、实际投入或花费的资金、耗用的资源,以及剩余的工期、资源和费用。

本章小结

软件项目管理是为了使软件项目能够按照预定的成本、进度、质量顺利完成，而对人员(People)、产品(Product)、过程(Process)和项目(Project)进行分析和管理的活动。软件项目管理的根本目的是为了让软件项目尤其是大型项目的整个软件生命周期(从分析、设计、编码到测试、维护全过程)都能在管理者的控制之下，以预定成本按期，按质地完成软件交付用户使用。

本章参考美国项目管理学会(PMI)提出项目管理框架，结合软件工程的实践，给出了软件项目管理的主要活动。软件项目管理的主要活动包括如下几个方面：软件项目计划、需求管理、进度管理、项目估算、风险管理、软件质量保证、软件配置管理等。

软件项目管理是软件工程和项目管理的交叉学科，是项目管理的原理和方法在软件工程领域的应用。与一般的工程项目相比，软件项目有其特殊性。首先，软件是纯知识产品，主要体现在软件产品的抽象性上，其开发进度和质量很难估计和度量，生产效率也难以预测和保证。其次，软件系统的复杂性也导致了开发过程中各种风险的难以预见和控制。因此软件项目管理的难度要比一般的工程项目管理的难度大，同时软件项目失败的概率也相对要高。

思考与练习

1. 什么是项目？现代工程项目的标准是什么？具有什么特征？

2. 项目管理与人类社会的发展息息相关，从三峡水利工程建设，到“神舟”载人航天伟大工程，再举例说明几个你所知道的大型项目工程，它对社会发展及进步起到了哪些重要作用？

3. 历史上传说的“巴比伦塔”倒塌的原因是什么？请举例说明沟通在现代项目管理中的重要性。

4. 国际项目管理学会(PMI)对项目管理的定义是什么？PMI 知识体系把项目管理划分为哪些知识领域？

5. 软件项目管理的 4P 指的是什么？为何说对 4P 进行分析和管理的活动，构成了软件项目管理的主要内容？

6. 项目管理者所做的决策会对软件工程项目组的效率和成败产生重大的影响。请列举一些实例来说明。

7. 参与软件项目的人员可以分为几类？各类人员的职责是什么？

8. 什么是软件范围？如何定义？

9. 哲学中还原论的基本思想是什么？它对于问题分解有何指导意义？

10. 试分析在项目管理三角形中，范围、时间、成本 3 个因素之间的关系及其互相影响。

11. 软件项目管理涉及的范围覆盖整个软件工程过程，它所包含的主要活动有哪些？

12. 现代需求工程一般被描述为 6 个步骤，请按照本章所介绍的“项目沟通的故事”，试

重新演绎图 3-6 中的内容,并说明其中的道理。

13. 一个规范的软件项目计划应包括哪些基本内容?

14. 什么是工作分解结构(WBS),如果你计划利用暑期完成一个社会调查,请设计你的 WBS。

15. 人员与工作量分配(项目进度)之间有何约束关系?请举例说明。

16. 常用的项目管理软件有哪些?请以微软公司的项目管理软件 Project 2003(或 Project 2007)为例,说明该软件所包括的功能。

17. 某软件组织准备研制一个 ATM(自动取款机)的嵌入式软件,试完成以下任务:

(1) 根据项目目标,制定一个较为详尽的软件项目计划。

(2) 请根据项目的范围,确定该软件的工作范围。

(3) 根据项目范围,确定并完成工作分解结构(WBS),并画出层次结构的任务分解图。

(4) 根据 WBS 的结果,用 Office Visio 初步制定项目进度图表。

第4章 软件项目估算

如果能对所说的内容进行度量，并能用数字表示之，那么表明你对所说的内容有所了解；反之，如果不能对所说的内容进行度量，或者不能用数字表示之，则表明你对它知之甚少，无法令人满意；也许你对它的认识刚刚开始，远未达到科学的程度。

——Lord Kelvin（英国著名物理学家）

4.1 软件项目估算概述

4.1.1 什么是估算

在现实生活中我们为许多活动做过估算，如计划驱车到目的地将要花费的时间，为某次旅行预算金额，估算房屋装修的费用等，这些都是进行估算的例子。现实生活中估算具有以下一些属性。

（1）为将来的事件做估算。你可以估计下次旅行所需要的时间和开销，但不用估计上次旅行花费的时间和开销，因为你已经知道了，这已成为历史事件。

（2）估算总是在没有完整信息的情况下做出的。因为是估计将来的事件，显然没有绝对的确定性，在真实事件发生时，事情可能会出差错。例如，当你估算一次旅行时间时，没有办法知道旅途中可能发生的交通阻塞或者其他影响行程的事件。

（3）每个估算都基于一定的假设。例如，当你驱车到某地时，会考虑到时间、天气、路线、交通条件等这样的假设。每个估算必须放在所做假设的环境中进行理解才有意义。所以，事前估算所有隐含的假设（可能发生的事件）会使某个过程或事情更为顺利地进行。

（4）做估算的目的是为了预测某些资源的需求。在任何情况下，估算是要弄清楚要完成某件事情或达到某个目标所需要的资源及数量。资源可能是时间、金钱、需要付出的人力等。于是，可以为估算给出以下定义[1]：

估算（估计）是一个设定期望的过程，它形成了基于某些明确阐述的假设来确定实现某些目标所需的资源数量的基础。

---

① 普林茨.软件工程.金维克，译.北京：科学出版社，2005

4.1.2 软件项目估算的特点

规划一项工作的精确度依赖于对它的了解程度。工程制造和建筑项目的成功实施有一个重要的基础，即要准确地对产品规模进行估算和规划。建筑就是一个很好的例子，在给我们提供一个估算之前，有经验的建筑商会取得我们想要的那类建筑物的详细信息，接着依靠他们之前的经验，算出建筑材料和劳动力的成本。例如，每平方米需花费多少钱来装饰，多少钱进行木工工作，多少钱涂泥灰和刷油漆等，经验丰富的施工人员对大工程所做的成本估算通常只有2%～5%的误差。

软件项目估算指估算软件开发过程中所花费的工作量及相应的代价。软件成本预测的直接目的是估算出成本和工作量，而其最终目的是为项目的过程和结果提供概率意义上的预测，同时可以用来改进软件过程。

不同于传统的工业产品，软件的成本不包括原材料和能源的消耗，主要是人的劳动力(生产力)消耗。另外，软件也没有一个明显的制造过程，它的开发成本是以一次性开发过程所花费的代价来计算的。因此，软件开发成本的估算，应是从软件计划、需求分析、设计、编码、单元测试、集成测试到认证测试，以整个开发过程所花费的代价作为依据。

软件生产是一种智力劳动，是资金密集、人力密集型产业。对于大型软件来说投入的人力多，周期长，其开发过程依赖于大量的、复杂的、高强度的脑力劳动，因此其开发成本是相当昂贵的，成本估算是其进行软件项目管理和控制的重要依据。成本和工作量估算是软件度量中发展较为成熟的一个分支，但即使如此，在预测的准确度和简便程度上面，也远未达到人们的期望程度。软件成本估算是软件成本管理和软件管理的核心任务之一，同时也是提高软件能力成熟度等级的一个重要的技术。

软件项目与其他工程项目相比，有其特殊性，主要表现在软件开发是处在动态开发环境下，受诸多可变要素影响的不确定性活动。因此，要在项目早期做到精确地估算工作量是不可能的。但是，与单凭经验猜测相比，能够考虑影响工作量估计的各种因素如系统规模和复杂性、开发人员的能力和经验、硬件的限制、软件工具和方法的应用等，采用恰当的估算技术可以大大提高估算的准确度。

4.1.3 软件项目估算的复杂性分析

软件工程专家 Roger S. Pressman 指出，软件成本及工作量估算永远不会是一门精确的科学，人员、技术、环境、策略都会影响开发所需的工作量及软件项目的最终成本。软件成本估算经过几十年的发展，已经取得了较大的成就，但其估算准确度相比其他工程行业，还有较大差距。

成本估算之所以难以准确，其原因主要是传统行业的任务可重复性在软件行业几乎不存在。传统行业的工程师可以利用以往项目的经验和专业知识进行预测，而软件产品由于是一个思维的智慧结晶，所以在很大程度上不如传统行业产品那么直观。另外，由于软件技术的发展速度极快，软件产品所面临的应用领域、软硬件环境及支持工具和应用问题的千差万别，导致上一个软件项目与下一个软件项目可能完全不同。

通常，我们完成一项任务时，需要根据其复杂性、工作量以及进度要求安排人力，但是软

件的工作量是很难估计的。一方面,软件开发实际上是抽象的逻辑思维过程,在写出程序并实际运行之前,软件开发的进度难于衡量,质量也难于评价,因此其工作量很难估计。另一方面,软件规模和复杂性是呈指数增加的,开发一个大型软件系统,往往需要成百上千人分工协作。由于软件系统结构复杂,各部分联系密切,大量的沟通、管理工作增加了工作量,给软件项目管理带来难度。团队开发软件虽然增加了开发力量,但也增加了额外的工作量。因此,在基于过程的软件项目管理中,合理有效的成本估算将有利于软件项目的管理和成本控制,提高管理效果。

软件开发过程当中需求和设计发生变更的可能性之大是传统行业所不能比拟的,开发人员面临的难以预测的问题更多,而这种应用的复杂性为项目过程导致了更多的不确定性。在技术问题之外的其他问题会导致预测结果受到操纵,如为了投标,在选择参数时强制与某些主观期望值匹配,而不是根据客观问题选择参数值。

软件开发项目超支、延期和不能满足用户的需求,已成为软件开发组织普遍存在的问题。根据 1999 年 Standish Group[①] 对当年美国项目的统计数字表明,只有 26%的项目是真正成功的,28%的项目是彻底失败的(即中途夭折的项目),介于两者之间是完成了的,但"受到质疑的"项目占 46%,这些项目被定义为存在费用超支、超出工期的项目。Standish Group 2003 年公布的调查数据中,在被调查的 1.35 万个项目中,绝对成功的项目比例远远低于 50%,仅为 34%;彻底失败的项目为 15%;受到质疑的项目占所有 IT 项目的 51%。

造成软件项目超支和延期的原因很多,不准确的估算就是其中一个重要原因。如果在做项目计划时,对项目的规模、工作量、成本和进度的估算不准确、不现实,那么即使项目管理、控制得再好,也很难达到预期的目标。软件项目估算不准确会导致一系列问题,如项目经理将工作量和成本估算得太低,可能会导致开发后期资源不足,进而影响软件产品的质量;如项目经理将工作量和成本估算得太高,则会影响资源妥善分配,甚至使管理层放弃开发有潜在收益的系统,丧失潜在的获利机会。因此,无论低估还是高估都会产生不利的影响。

在大多数软件开发项目中,大量的成本和主要的进度一般都花费在项目的软件产品定义、需求规约、软件设计、软件实现、软件测试和交付中。此外,在项目的初始规划阶段,经常只了解产品最一般的特性,而对于大多数项目而言,初始成本和进度估算必须基于产品的最一般特性。这些特性基本都在第一个主要工作产品——软件需求规约中给出。所以,在软件需求规约被确认之后,应重复进行一次估算和制定进度的过程。图 4-1 给出了在产品开发的各个点上有关成本估算中可能出错的范围[②]。在软件概念的开发阶段(产品定义与可行性阶段),估算可能低于 4 倍或高于 4 倍(即 4X 到 0.25X)。换言之,总的范围为 16∶1。之后,在设计阶段其范围的每一边就窄了 1.5X,总的范围为 2.25∶1。对于那些重复(或多或少)了以前一些过程的项目而言,由于使用稳定的技术和人员来开发类似的产品,因此这些项目就应接近该区域的中间(最好的预测)。而对于那些具有新意的产品,由于使用了一些不稳定的人员、不熟悉的技术,那么对这样产品的估算就具有很大的变化性。

---

① Standish Group 是美国专门从事跟踪 IT 项目成功或失败的权威机构,在它每年的 CHAOS Report 报告中给出了 IT 项目相关调查数据结果。

② Boehm B et al. Cost Models for Future Life Cycle Processes: COCOMO 2. Annals of Software Engineering,1: 57~80

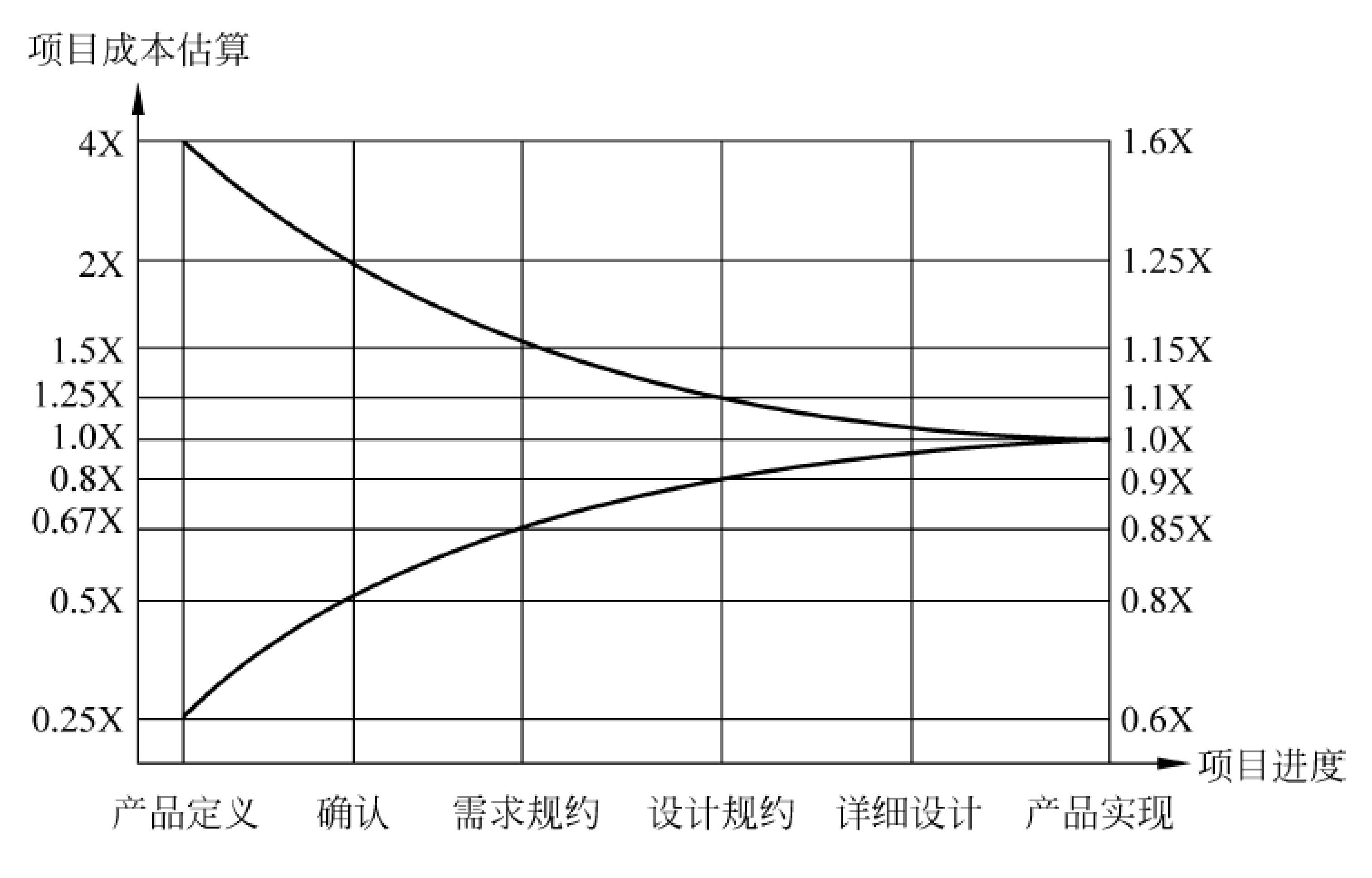


图 4-1 项目估算中的错误范围

“不管宇宙从何状态开始，它都必须结束于一个高度有序的状态，早期宇宙有可能处于无序的状态，这意味着无序度将随时间减小。”[①]项目组织与项目估算也是如此。如图 4-1 所示，软件成本估算的准确性随着软件项目生命周期的发展越来越高，这是因为在软件开发阶段初期，与软件产品相关的很多因素都还不确定，因此软件估算成本的变动范围系数为 4X。随着软件项目向后期发展，各种相关的信息和数据将越来越明确和清晰。例如，当完成了可行性分析并确定了产品定义以后，成本估算的变动范围就降到了 2X。随着软件项目往后面阶段的发展，越来越多的项目特性得到确认，估算的准确性也就相应提高了。

4.1.4 软件项目估算的相关内容

一般而言，软件项目估算首先是规模估算。有了规模的估算值，通常也就能采用某种公式来获得工作量的估算值了。另外对于项目管理者和相关利益方而言，软件成本和项目持续的时间是项目开发成功的主要关键因素。所以，软件项目估算主要包括以下所述内容，如图 4-2 所示。

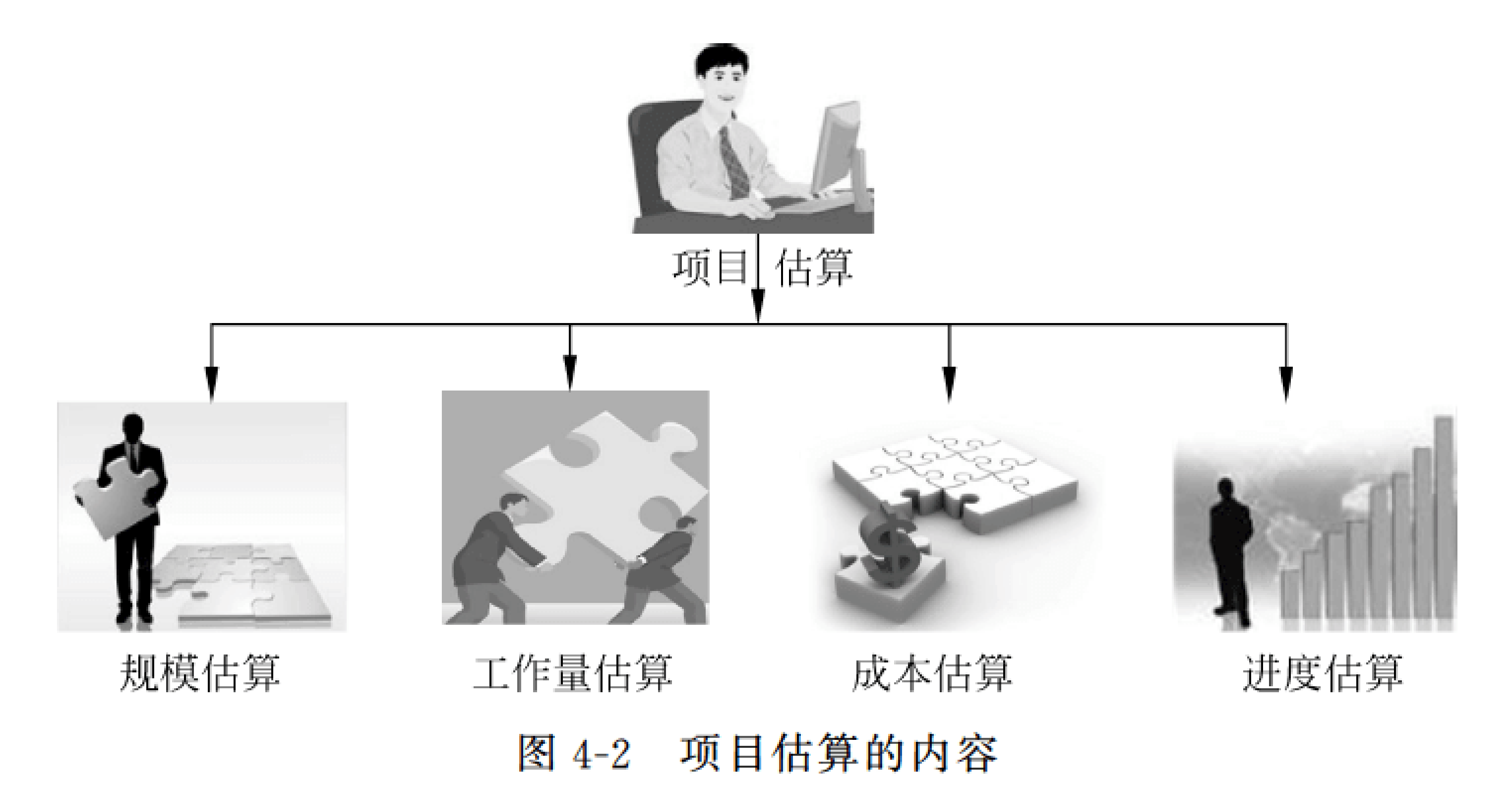


图 4-2 项目估算的内容

① 史蒂芬·霍金. 时间简史. 许明贤，译. 湖南：湖南科学技术出版社，1994：133

1. 规模估算

所要开发的软件的规模是一个需要首先做出估算的参数，它是估算工作量的基础，有了这个数据才可以对工作量和进度做出估算。进行规模估算应该十分了解项目的需求，一个项目的需求对估算规模来说应是有界的。软件规模的衡量尺度有代码行、功能点数量、对象点数量等。

2. 工作量估算

工作量估算是对开发软件产品所需的人力的估算，它是任何软件项目所共有的主要成本。它和进度估算共同决定了开发团队的规模和构建，通常以人天、人月、人年的形式来衡量，并且有转换系数在不同单位之间进行转换。工作量估算是由规模和与项目有关的因素所驱动的，如团队的技术和能力、所使用的编程语言和开发平台、平台的可用性与适用性、团队的稳定性、项目中的自动化程度等。

3. 成本估算

在任何软件项目中，成本中的一个主要组成部分是人力成本（基于估算出的工作量）。此外，其他成本费用也需要做出估算，它们和人力成本一起用于估算项目的总成本。

4. 进度估算

进度是项目开始日期到项目结束日期之间的一个时间段，进度估算是项目计划和控制的基础。进度估算是基于里程碑的，如各个阶段的结束。

4.2 项目规模估算

众所周知，用“字节”可以表示计算机存储设备的容量，用“平方米”可以衡量住房建筑面积大小。然而，长久以来，软件产品的规模度量却是个争论不休的问题。软件项目的规模估算历来是比较复杂的事，因为软件本身的复杂性、历史经验的缺乏、估算工具缺乏以及一些人为错误，导致软件项目的规模估算往往和实际情况相差甚远。因此，估算错误已被列入软件项目失败的四大原因之一。

在规模估算之前，软件功能需求必须被定义。在项目早期定义需求可能是非常困难的任务。然而，在对需求一无所知的情况下，精确地估算出项目的成本和进度是不可能的。如果知道部分需求，那么估算基于已知的需求，并且使每一个人都清楚目前的估算仅仅是基于那些已知的需求；如果使用了增量或演进的开发策略，那么估算基于增加的已定义需求。

表 4-1 给出了一些大型软件项目（产品）的源代码行数（Line of Code，LOC），由于我们对这些产品的特性与功能具有一定的熟知程度，所以可以比较直观地感受到类似产品的规模。

表 4-1 大型软件项目(产品)的规模(LOC)

软件性质	工作量(人年)	规模(KLOC)	备注
编译语言			开发时间
-Pascal,C	10(人年)	20～30(KLOC)	1～2 年
-Cobol,Fortran	80～100(人年)	100～200(KLOC)	2～3 年
-Ada	150～200(人年)	>300(KLOC)	
关系数据库			开发时间
Oracle,DB2,…,	300～500(人年)	300～600(KLOC)	3～5 年
大型实时控制系统			
-航天飞机	>1000(人年)	2200(KLOC)	用 HAL 写成,耗时 6 年
-SAPEGUARD	5000(人年)	2260(KLOC)	美国 20 世纪 70 年代开始研究的反弹道导弹防御系统
-Sabre	955(人年)	960(KLOC)	由 IBM 公司研制的美国航空订票系统,用 PL/1 语言写成,耗时 7 年
操作系统,设计系统	2500～5000(人年)		
-MVS,VMS		5000～10000(KLOC)	MVS:多重虚拟存储 VMS:虚拟网络服务
-Windows NT 1.0		4000～10000(KLOC)	1992 年发布
-Windows XP		20000～30000(KLOC)	2001 年发布,内核采用 C/C++ 编程实现
-Windows Vista		30000～40000(KLOC)	2005 年发布,开发时间为 6 年,耗资 30 亿美元
大型应用软件			发行时间
-Project for Windows 3.0		248(KLOC)	1992
-Word for windows2.0		326(KLOC)	1991
-MS Excel 4.0		851(KLOC)	1992
人工智能软件			一般用 C 语言编程实现
-Lisp,Prolog	10～20(人年)		
-专家系统	20～30(人年)		

表中主要数据来源于:普林茨.软件工程.金维克,译.北京:科学出版社,2005

表中的这些数字看起来似乎有些抽象,但如果将它们与日常生活中的事物联系起来,就能更清楚地了解它们所代表的极端复杂性。例如,一个 100KLOC 的软件,包括文档及附件在内,其规模差不多相当于 10 本 400 页的图书。另外,估算值并不是一个简单的人月或人年数字,例如,10 个人年的工作量,不可能要一个人干 10 年来完成。

4.2.1 基于代码行的规模估算

LOC 指所有的可执行的源代码行数,包括可交付的工作控制语言语句、数据定义、数据类型声明、等价声明、输入/输出格式声明等。代码行数量是对软件源代码行数的直观测量,由于代码行数量是所有软件开发项目都有的“产品”,且容易精确估算甚至自动计算获得,在目前仍然是常用的软件规模度量方法。

代码行数量和人月均代码行数可以体现一个软件生产组织的生产能力,可以根据对历

史项目的审计来估算项目的规模。

LOC 方法的优点如下。

(1) 代码行度量方法的最大优势在于它容易获取。通过对项目的代码行的“粗略数量级”的测算，并且与历史项目的代码行数量进行对比，就能够很快地对当前项目的规模和工作量有一个比较真实的理解。

(2) 代码行数是比较公认的软件规模的标准。当代码行数量与其他关于项目开发复杂度、开发团队能力水平等因素结合起来以后，可以提供足够的信息用于软件的测算，容易在项目各利益方中取得共识。

代码行规模度量的方法缺点也是非常明显的，主要有以下几个。

(1) 很难在项目分析论证和计划阶段就对软件的规模进行相对准确的度量。

(2) 即使进行度量，也必须依靠于评估人员的经验所得。虽然只要评估人员经验足够，就可以获得比较准确的度量结果，但是这对评估人员的能力要求比较强，并且难以由第三方对评估人员的工作偏差做出修正。

(3) 不同软件项目使用的技术不一样，这一点非常影响软件规模的度量。例如，同一个功能，使用 Java 语言和使用 C 语言所涉及的代码行可能相差数十行，甚至数百行。即使同为 Java 语言，使用不用的框架所需要编写的代码行也不一样。

总体来说，LOC 方法很适合软件业初期，对使用比较简单的程序开发语言的项目进行规模度量。但是随着软件技术的发展，使用 LOC 方法进行度量的难度越来越大，其准确和可操作性在逐渐降低。

4.2.2 功能点估算

1. 什么是功能点

功能点(Function Point，FP)测量是在需求分析阶段基于系统功能的一种规模估计方法。该方法由 IBM 的工程师 Allan Albrech 于 20 世纪 70 年代提出，随后被国际功能点用户协会(The International Function Point Users'Group，IFPUG)提出的 IFPUG 方法继承，是基于应用软件的外部、内部特性以及软件性能的一种间接的规模测量。其特征是：“在外部式样确定的情况下可以度量系统的规模”，“可以对从用户角度把握的系统规模进行度量”。通过研究初始应用需求来确定各种输入、输出、计算和数据库需求的数量和特性。如表 4-2 所示。

(1) 外部输入：输入就是用户借以添加或修改数据的屏幕或表格。

(2) 外部输出：输出就是程序产生的给用户看的屏幕或报表。注意，输出需要对计算单元进行单独处理；例如，在一个工资应用程序中，100 张输出的工资清单仍然算一个输出。

(3) 外部查询：查询是一种联机输入，它导致软件以联机输出的方式生成某种即时的响应。每一个不同的查询都要计数。

(4) 内部逻辑文件：数据文件是应用的修改或者保存的逻辑记录的集合。例如，一个数据文件可以是磁盘文件，也可以是关系数据库中的一张表。

(5) 外部接口文件：接口就是与其他应用共享的文件，包括输入和输出的磁盘文件、共享数据库、参数列表等。

估算出表 4-2 所示的每类功能点的数量后，再对功能点的复杂性(用加权因子表示)做

出判断，分成简单、一般、复杂3种情况，初始功能点数（Initialization Function Points，IFP）是通过表4-2中的功能总数量×加权因子计算出来的。

表4-2　初始功能点估算表

功能点类别	数量	加权因子			初始功能点数（数量×加权因子）
		简单	一般	复杂	
外部输入		3	4	6	
外部输出		4	5	7	
外部查询		3	4	6	
内部逻辑文件		7	10	15	
外部接口文件		5	7	10	
总计数值					

2. 功能点的复杂度调整因子

通过对上面几个重要信息域的收集，可以定义出软件项目的初始功能点的数量，再使用如式(4-1)所示的关系式：

$$FP = 总计数 \times [0.65 + 0.01 \times SUM(F_i)] \tag{4-1}$$

其中：总计数是由表4-2所得到的所有加权计数项的和；$F_i(i=1\sim14)$是对影响产品规模的14个因素进行分析确定的"复杂度调整值"，取值范围是0～5，它们应通过逐一回答表4-3所提出的问题来确定；$SUM(F_i)$是求和函数。式(4-1)中的常数和应用于信息域计数的加权因数可根据经验确定，从而计算出调整后的FP数量。

表4-3　功能点的复杂度调整值

F_1	数据通信	F_8	在线升级
F_2	分布式数据处理	F_9	复杂处理
F_3	性能	F_{10}	可重用性
F_4	资源需求	F_{11}	易安装性
F_5	事务频率	F_{12}	易操作性
F_6	在线数据输入	F_{13}	多点运行
F_7	终端用户效率	F_{14}	易变更

对于以上的每一个影响因子，根据其影响程度定义为如表4-4所示的5个等级。

表4-4　影响程度级定义

0	1	2	3	4	5
没有影响	偶有影响	轻微影响	一般影响	较大影响	严重影响

根据式(4-1)可以知道，复杂度调整因子的值在0.65～1.35范围内。

3. 功能点分析方法的不足

有学者认为功能点分析方法存在以下一些缺点。

(1) 通常很难确认一个应用软件的部件。例如，什么是逻辑文件？目前还没有一个清

晰的定义来进行判断。

(2) 为每个功能点的部件赋予了一个加权值，而这种赋值的方式尚有待商榷，有时还不是很确定。

(3) 上述两个问题同样也存在于复杂度调整因子的权重设定中。

(4) 没有提供内部复杂度计算的含义。因此，在分析时对算法和其他一些情况会造成分歧，无法得到统一的答案。

4. 功能点转换为源代码行数

功能点和源代码行是从两个不同的角度来度量软件规模，它们之间存在着较强的相关性。对于具体的软件开发部门，可根据该部门的历史数据经过统计处理获得功能点数和源代码行之间的关系。表4-5给出了在不同编程语言环境下每个功能点对应的源代码行数的参考值。

表4-5 不同编程语言下FP与LOC间换算关系

编程语言	LOC/FP	编程语言	LOC/FP
Java	53	UNIX Shell Scripts	107
Visual C++	34	Lisp	64
Visual Basic	29	4GL	20
PowerBuilder	16	Pascal	91

数据来源：Barry W. Boehm. Software Cost Estimation With COCOMO Ⅱ.

4.2.3 基于计划评审技术的规模估算

计划评审技术(Program Evaluation and Review Technique,PERT)是20世纪50年代，美国国防部为发展北极星导弹计划而研究出来的方法，是对不确定性高的项目进行估计而采用的方法。PERT使计划中工作与工作之间的逻辑关系肯定，但是由于每项工作的持续时间不肯定，PERT一般采用加权平均时间估计，并对按期完成项目的可能性做出评价。

PERT规模估计技术是一种基于β分布和软件各部分单独估算的技术。在应用该技术时，对于每个软件部分要产生3个规模估算量，分别表示乐观、可能与悲观的值，所以又称为三点估算法，如图4-3所示。

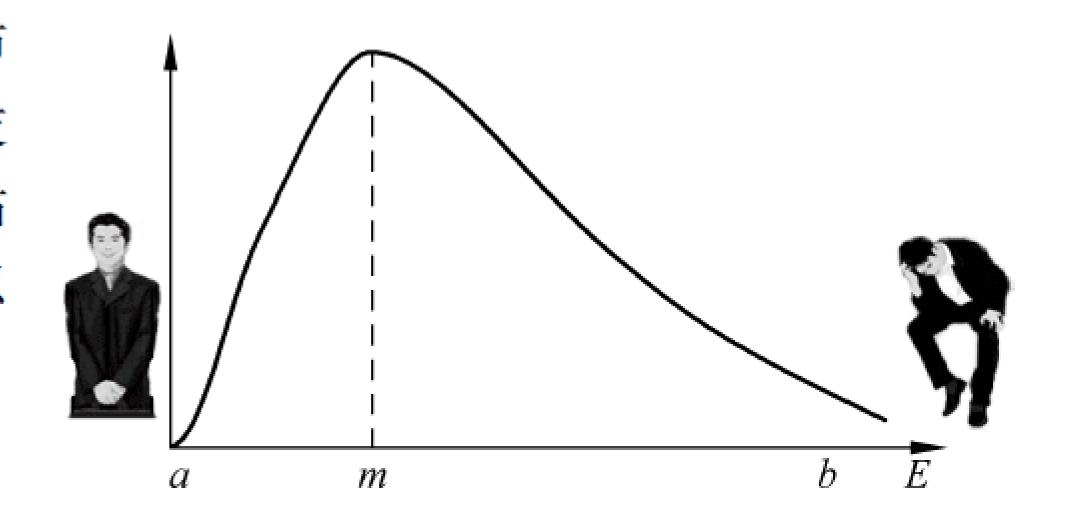


图4-3 基于规模的三点估算方法

a_i：软件第 i 部分可能的最低规模。

m_i：软件第 i 部分最可能的规模。

b_i：软件第 i 部分可能的最高规模。

则第 i 部分规模期望 E_i 和总的软件系统规模期望 E 为：

$$E_i = (a_i + 4m_i + b_i)/6 \tag{4-2}$$

$$E = \sum_{i=1}^{n} E_i \tag{4-3}$$

为描述估算的样本数据与期望值的距离，这里引入样本方差的概念。在概率统计中，样本中各数据与样本平均数的差的平方和的平均数叫做样本方差。

在三点规模估算技术中，计算第 i 个部分的方差公式为：

$$\sigma_i^2 = \left(\frac{b_i - a_i}{6}\right)^2 \tag{4-4}$$

方差的计量单位和量纲不便于从经济意义上进行解释，所以在实际统计中多用方差的算术平方根——标准差来测度统计数据的差异程度。标准差用来反映一组数值与平均值的离散程度。标准差数值越大，代表大部分的数值和其平均值之间差异较大；标准差数值越小，代表这些数值较接近平均值。

在三点规模估算技术中，标准差的表示公式为：

$$S = \sqrt{\sum_{i=1}^{n} \sigma_i^2} \tag{4-5}$$

样本方差和样本标准差都是衡量一个样本波动大小的量，样本方差或样本标准差越大，样本数据的波动就越大。在 PERT 估算中，它表明估算的不确定性程度。如果方差很大，则说明悲观和乐观估值相差太多，这表明完成该工作存在的不确定因素较多；反之，则表明不确定因素较少。

4.3 工作量估算

工作量的含义是完成一个任务所需要的人力与时间。项目经理把工作量分配给具体的工程师，把工作量分布在详细的项目计划中，这就是依据工作量所进行的项目管理。所以，工作量提供了项目管理的基础。

软件规模（代码行和功能点）估算出来后，那它跟我们的工作量有什么样的联系呢？一般而言，工作量是软件规模（KLOC 或 FP）的函数，软件估算模型使用由经验导出的公式来预测软件开发工作量，工作量的单位通常是人月（Person Month，PM）。

另一个与工作量相关的概念是生产率，在制造业中可以通过统计生产的产品数量，然后除以人数和工作耗费的小时数算出生产率。对于软件开发的生产率而言，程序员每人每月源程序代码行数（或功能点数量）是一种广泛使用的生产率度量指标。这个指标是将交付上来的源程序代码行数和功能点数，再用这个数量除以完成该项目总的人数。这个时间包括系统分析、设计、编码、测试和文档编写所用的时间。

软件生产率的另一种表示方法是建立在心理学研究基础上的。Ivan Steiner 提出一个软件实际生产率的概念，其模型可以简单地表述如下：

实际生产率＝潜在的生产率－不完善的过程带来的损失

潜在的生产率被定义为“个人或团体等在最大可能地利用他们的资源时所产生的最高水平的生产率”。它是任务属性（如产品的复杂性、数据库大小）和团体的资源（如个人能力、经验水平、软件工具）这两组因素的函数。由于不完善过程带来的损失是指由于沟通和协调费用的超支以及（或）过低的动机（激励）所导致的生产率的降低。

4.3.1 用代码行与功能点估算工作量

1. 代码行估算工作量

一种基本的软件资源规划的方法是收集某些前期项目的历史性生产率的数据。例如，如果所收集的数据显示平均每天可以产出100LOC左右，则可以估算6000LOC的项目大概要用60天。更进一步，如果历史数据显示生产率的正常波动在每天80～120LOC之间，则可以估算新项目的大概范围在50～75天之间。虽然这个范围看起来比较大，但它却是前期经验的准确反映。我们最好能知道这个范围，而不是简单地假设一个平均值。现在，如果项目耗时比所期望的时间要长，我们至少能估算出可能的规划。

【例4-1】 某软件公司统计发现该公司每10 000行C语言源代码形成的源文件(.c和.h文件)约为250KB。某项目的源文件大小为4MB，则可估计该项目源代码大约为16.4万行，该项目累计投入工作量为160人月，每人月费用为10 000元(包括人均工资、福利、办公费用等)，试计算该项目中每代码行(1LOC)的成本C和生产率P。

解：每代码行成本$C=(160\times 10\,000)/164\,000\approx 10$(元/LOC)

人均生产率$P=164\,000/160\approx 1000$(LOC/人月)

2. 用功能点估算工作量

根据行业经验数据，使用Java语言开发时，一个功能点的生产时间在1～1.4天内(此处的生产时间仅指软件开发活动，它通常包含需求分析、设计、编码、测试等活动，但不包含项目管理、软件维护等支持性且因项目要求的不同而差异较大的活动)。那么我们就可以根据计算出来的功能点推算出项目可能的人力。

项目总生产人力典型计算公式：项目总生产人力＝项目功能点数×生产率

在项目结束后来计算功能点，有助于我们根据实际的生产时间来计算单位内部的生产效率等度量指标。

生产效率典型计算公式：生产效率＝项目总生产人力/项目功能点数

例如，A软件项目的规模是100功能点，根据行业基准知道平均成本是5000元/功能点，那么本项目的成本预测就是50万元；又根据行业基准知道平均生产率为1功能点/人天，则计算得到项目需要投入100人天的工作量，这些计算的结果将成为软件项目管理的基础。

4.3.2 基于数学模型的工作量估算

采用数学模型这种方法是学术界热衷的，因为有数学公式的东西更显得有学术味道。这类方法适合于非常成熟的软件机构，该机构需积累丰富的历史数据，才能够归纳出数学模型来指导新项目的规划。

一个典型的估算模型是通过对以前的软件项目中收集到的数据进行回归分析而导出的。这种模型的总体结构具有如式(4-6)所示的数学模型：

$$E = A + B \times (ev)^c \tag{4-6}$$

其中，A、B和C是由经验导出的常数；E是以“人月”为单位的工作量；ev是估算变量，如代

码行(LOC)或者功能点(FP)。

由此提出了许多面向 LOC 的估算模型：

$E=5.2(\text{KLOC})^{0.91}$　　Walston-Felix 模型

$E=5.5+0.73(\text{KLOC})^{1.16}$　　Bailey-Basili 模型

$E=3.2(\text{KLOC})^{1.05}$　　Boehm 的简单模型

$E=5.288(\text{KLOC})^{1.047}$　　Doty 模型，在 KLOC>9 的情况下使用

同样，也提出了许多面向 FP 的估算模型。主要包括：

$E=-91.4+0.355\text{FP}$　　Albrecht 和 Gaffney 模型

$E=-37+0.96\text{FP}$　　Kemerer 模型

$E=-12.88+0.405\text{FP}$　　小型项目回归模型

从上述的模型中可以看出：每一个模型对于相同的 LOC 或 FP 值，会得出不同的结果(部分原因是因为这些模型一般都是仅从若干应用领域中相对很少的项目中推导出来的)。其含义很清楚，估算模型必须根据当前项目的需要进行调整。

4.3.3 COCOMO 模型

1. 什么是 COCOMO 模型

COCOMO 是英文 ConstructiveCostModel 的缩写，翻译为“构造性成本模型”。在项目度量领域中，COCOMO 是一个非常具有影响力的模型。该模型是由 Boehm 博士在 1981 年于《软件工程经济学》中提出的软件参数模型，可以用来估算软件成本、工作量和进度计划。

20 世纪 90 年代，软件项目管理和开发技术与工具发生了很大变化，为适应新的软件成本估算和过程管理的需要，1994 年 Boehm 重新研究和调整了原有模型，根据未来软件市场的发展趋势，发表了 COCOMO Ⅱ模型，反映了现代软件过程与构造方法。

(1) COCOMO 模型的思想和技术要点是：软件成本和工作量估算是一种实验性方法，一定要通过大量实验和已有软件项目的数据建立实验模型。

(2) 软件工作量估算要区分软件类型和软件的不同开发方式。

COCOMO 用以下 3 个不同层次的模型来反映软件项目不同程度的复杂性。

(1) 基本模型(Basic Model)：是一个静态单变量模型，它用一个已估算出来的源代码行数(LOC)为自变量的函数来计算软件开发工作量。

(2) 中间模型(Intermediate Model)：将软件开发工作量作为程序规模及一组“成本驱动因子”的函数来进行计算。其中，成本驱动因子包括对产品、硬件、人员及项目属性的主观评估。

(3) 详细模型：(Detailed Model)：包括中间 COCOMO 模型的所有特性，但用上述各种影响因素调整工作量估算时，还要考虑对软件工程过程中分析、设计等各步骤的影响。

2. 基本 COCOMO 模型的 3 种应用开发模式

根据不同应用软件的不同应用领域，基本 COCOMO 模型划分为以下 3 种软件应用开发模式。

(1) 组织模式(Organic Mode)。这种应用开发模式的主要特点是在一个熟悉稳定的环

境中进行项目开发,该项目与最近开发的其他项目有很多相似点,项目相对较小,而且并不需要许多创新。

(2) 中间应用开发模式 (Semidetached Mode)。这是介于组织模式和嵌入式应用开发模式之间的类型。一般来讲,信息系统是组织式的,而实时系统是嵌入式的。

(3) 嵌入式应用开发模式 (Embedded Mode)。在这种应用开发模式中,项目必须在严格的约束条件下开发,而且要求项目有很大的创新,要解决的问题无法借助于经验。

基本 COCOMO 模型的 3 个开发模式都采用如式(4-7)所示的形式:

$$E = a \times Size^b \tag{4-7}$$

其中:E(Effort)是以人月为单位的工作量;$Size$ 是以千源代码行(KLOC)计数的程序规模;a 和 b 是两个随开发模式而变化的因子。其取值如表 4-6 所示。

根据计算出的工作量,可以由式(4-8)计算所需的开发时间:

$$D = cE^d \tag{4-8}$$

其中:E 是以人月为单位的工作量;D 是以月为单位的开发时间;参数 c,d 按表 4-7 取值。

表 4-6　基本 COCOMO 模型的相关参数

软件项目	a	b
组织模式	2.4	1.05
半分离模式	3.0	1.12
嵌入模式	3.6	1.20

表 4-7　开发时间的参数取值

软件项目	c	d
组织模式	2.5	0.38
半分离模式	2.5	0.35
嵌入模式	2.5	0.32

基本 COCOMO 模型的估算准确性不够理想,但是却非常简单易用,项目管理者可以快速预测出软件的开发成本和进度等信息。

3. 中级模型

基本 COCOMO 模型能给出一个快速而简略的估计,但其结果的精度不够。它没有考虑到开发技术、人员水平等环境因素的变化。因此,Boehm 在中级模型中引入了 15 个成本驱动因素(Cost Drivers)来把软件开发的环境因素考虑进来。

中级 COCOMO 模型的表达式如式(4-9)所示:

$$E = a\text{KLOC}^b \times \text{EAF} \tag{4-9}$$

其中:E 是以人月为单位的工作量;KLOC 是估算的项目代码行数;系数 a 和指数 b 的取值同表 4-6;EAF(Effort Adjustment Factor)即成本驱动因子,共有 15 个,可将其分为产品属性、计算机属性、人员属性和项目属性 4 类,其典型取值是 0.9～1.4。

【例 4-2】 现要开发一个规模为 100KLOC 的微处理器上的嵌入型电信处理程序,试使用 COCOMO 模型计算所需的工作量和开发时间。

解:① 使用基本 COCOMO 模型:

开发工作量 $E=3.6\times100^{1.2}=904$(人月)

开发时间 $D=2.5\times904^{0.32}=22$(月)

② 使用中级 COCOMO 模型:

开发工作量 $E=2.8\times100^{1.2}=703$(人月)(取式 4-9 中的 a 为 2.8)

开发时间 $D=2.5\times703^{0.32}=20.4$(月)

如果参加分析与设计人员的月平均工资为 6000 元，按基本 COCOMO 模型计算，则开发该项目的人员工资支出成本为：

$$工资成本=6000\times904=542(万元)$$

4. 详细 COCOMO 模型

详细 COCOMO 模型包含中间 COCOMO 模型的所有特性，还包含了"成本驱动"对软件工程过程每一步(需求分析、概要设计、详细设计、编码与单元设计、集成测试和维护)的影响。

详细 COCOMO 模型对项目的每一个阶段给出不同的成本驱动因子分级表。在实际的开发中，从产品设计到集成测试称作开发阶段。需求分析和维护阶段不同于其他 4 个开发阶段的处理方式。详细 COCOMO 模型为每一个成本驱动属性提供了一组针对不同阶段的工作量因子，用这些因子来决定完成每一阶段所需的工作量。

针对每一个影响因素，按模块层、子系统层、系统层，有 3 张工作量因素分级表供不同层次的估算使用。每一张表中工作量因素又按各个不同阶段给出。在许多部件的成本因素级别完全相同的情况下，详细 COCOMO 模型还提供一个 3 级产品层次结构来处理不必要的重复。此层次结构将有些随底层各模块变化而变化的效应放在模块层处理，将有些不经常变化的效应放在子系统层处理，有效效应(如产品总规模的效应)放在系统层处理。

4.3.4　COCOMO Ⅱ模型

1990 年后，软件项目管理和开发技术与工具发生了很大的变化，出现了快速应用开发模型、软件重利用、再工程、CASE、面向对象方法以及软件过程成熟度模型等一系列软件工程方法和技术。早期的 COCOMO 模型已经不再适应新的软件成本估算和过程管理的需要，因此，1994 年 Boehm 重新研究和调整原有模型，根据未来软件市场的发展趋势，发表了 COCOMO Ⅱ。从初始的 COCOMO 模型到 COCOMO Ⅱ的演化反映了软件工程技术的进步。

1. COCOMO Ⅱ模型的改进

正像 COCOMO 模型中描述的那样，将来的软件项目必须适合特定的软件过程驱动因素的需要，满足软件重利用、用户需求理解层次、市场和进度限制、可靠性要求。

COCOMO 模型的演化反映出软件工程技术在过去的 20 年间不断走向成熟，在现代软件工程研究成果的基础上，它将未来的软件市场划分为基础软件、系统集成、程序自动化生成、应用集成、最终用户编程 5 个部分。COCOMO Ⅱ的主要变化如下。

(1) COCOMO Ⅱ使用 3 个螺旋式的生命周期模型：应用组合模型、早期设计和后体系结构。

(2) 使用 5 个规模因子计算项目规模经济性的幂指数 B，代替了原来按基本、中间和详细 COCOMO 模型的不同使用固定指数的方法。

(3) 扩展功能点测量，使用源代码行(SLOC)代替 DSI(源指令条数)。

(4) 新增加成本驱动因子，删除部分不适用的成本驱动因子。

(5) 改变原有成本驱动因子的参数值，如软件再工程和代码自动转换等，以适应当前的

软件测度技术。

COCOMO 模型在不断演化。从 COCOMO Ⅱ的变化可以看出，它吸收了对象点、功能点、能力成熟度模型等其他软件工程研究成果，增加了模型的灵活性、可扩展性、可操作性，表现出强大的生命力。

2. COCOMO Ⅱ的 3 个生命周期模型

最新的 COCOMO Ⅱ与软件生命周期相结合，通过 3 个不同的模型分别对 3 个不同的阶段进行估算。各个生命周期阶段模型对项目的理解深度不同，因此能够度量的成本驱动因子也就不同。

(1) 应用组合模型(Applications Composition Model)。应用组合模型是用来度量采用集成化计算机辅助软件工程(ICASE)环境进行组件组装式开发的项目。应用组合模型是基于对象点(Object Point)的度量模型，它通过计算屏幕、报表、第三代语言(3GL)模块等对象点的数量来确定基本的规模，每个对象点都有权重，由一个 3 级的复杂性因子表示，将各个对象点的权值累加起来得到一个总体规模，然后再针对复用进行调整。

(2) 早期设计模型(Early Design Model)。早期设计模型应用于软件项目的初期，这时我们对于将要开发的产品的规模，目标平台的环境，参与项目人员的能力，或将使用的过程的细节特征都不清楚。作为成本的初步估算，可应用于应用生成器，系统集成和底层开发部分。COCOMO Ⅱ模型使用功能点和等价代码行进行规模估算，为此给出了包含 7 个成本驱动因子的估算模型。

(3) 后体系结构模型(Post-architecture Model)。就像它的名字所言，在这个阶段，软件结构经历了最后的构造、修改，并在需要时开始创建要执行的系统，相关信息不断细化。关于成本驱动输入的详细情况已经知道，所以估算较为精确。因此 COCOMO Ⅱ的后架构阶段估算考虑了 17 个成本驱动因子。这些精心设计的成本因子一方面体现出产品、硬件、人员、项目因素对软件开发工作量的影响，另一方面也考虑到了可度量性和易用性。

COCOMO Ⅱ模型代表了软件估算的一个综合经验模型。功能点估算过程本身带有一定的主观性，COCOMO Ⅱ模型的计算公式中也有不少需要主观估计定值的参数。这些都会导致估算结果与实际情况有所偏差。

3. 早期设计模型和后体系结构模型的工作量估算

COCOMO Ⅱ模型的最大特色在于其成本驱动因子对工作量的调整，扩展了基本模型。项目开发者需要确定各个驱动属性的级别，早期设计模型和后体系结构模型的工作量估算使用同样的规模估算方法和规模驱动因子。对于二者的工作量公式如式(4-10)所示：

$$PM = A \times Size^{B} \times \prod_{i=1}^{n} EM_i \tag{4-10}$$

其中：PM 是工作量(人月)，人月是指除去节假日之后一个人在一个月内所完成的项目工作量；常数 A 通常取值为 2.94；B 反映了项目的规模经济性，当它大于 1 时所需的工作量(PM)的增加速度大于软件规模($Size$)的增加速度，体现出规模非经济性，反之，B 小于 1 则表示规模经济性；EM_i 是从公认的值表中得到成本驱动因子的值，所有成本驱动因子的乘积就是工作量调整值 EAF(Effort Adjustment Factor)；对于早期设计模型，$n=7$；对于后

体系结构模型，$n=17$。

关于 COCOMO Ⅱ模型的详细定义与应用可参考 Boehm 所著的《软件成本估算：COCOMO Ⅱ模型方法》一书。

4. COCOMO Ⅱ的应用组合模型——对象点分析方法

COCOMO Ⅱ的应用组合模型通过计算屏幕、报表、第三代语言组件的对象点(Object Point)数来确定一个初始的规模测量。对象点分析方法是一种相对较新的软件规模估算方法，与面向对象的开发方法中所述的“对象”的含义并不相同，在对象点分析方法中的术语“对象”指的是屏幕、报表、编程模块等事物。

对象点估算过程如下。

(1) 得到对象的数目：估计构成这个应用的窗口(Screen)、报告(Report)和 3GL(第 3 代编程语言)构件的数目。

(2) 依据特征：将这些对象实例分类成简单、中等、困难 3 个复杂等级。用权值表示要实现该复杂度水平上的一个实例所需要的相对工作量，即权重。具体定义如表 4-8 所示。

表 4-8 对象点复杂度权重

对象类型	复杂度权重		
	简单的	中等的	复杂的
屏幕	1	2	3
报表	2	5	8
3GL 构件			10

(3) 决定对象点：累加所有带权重的对象实例，从而得到对象点的数目。

(4) 估计项目中能达到的复用的百分比 REUSE，计算将开发的新对象点数目 NOP (New Object Point)。

$$\text{NOP} = \text{对象点数目} \times (100 - \text{REUSE}\%)/100 \qquad (4\text{-}11)$$

(5) 根据表 4-9 计算出生产率(PROD)。

$$\text{PROD} = \text{NOP}/\text{人月} \qquad (4\text{-}12)$$

表 4-9 对象点工作量转换

开发人员的经验和能力	非常低	低	正常的	高	非常高
生产率(PROD)	4	7	13	25	50

(6) 然后通过将 NOP 除以 PROD 计算出执行项目需要的人月数的估计。

$$\text{PM} = \text{NOP}/\text{PROD}$$

当存在可以利用的现成构件时，其中的一些对象就可能不需要开发。对象点得分要根据这一点进行调整。

【例 4-3】 假定一个应用程序包含 600 个对象点，30%可以通过使用现成的构件来提供，那么调整后的新的对象点(NOP)是多少?

解：NOP＝600×(100－30)/100＝420

假定开发环境的生产率是正常的(取表 4-9 中的生产率为 13),那么项目的估计工作量将是 420/13=32(人月)。

4.3.5 Putnam 模型

1978 年,Putnam 提出了大型软件项目工作量(一般在 30 人年以上)估算模型。它是一个动态多变量模型,揭示了软件项目的工作量、软件开发时间和程序代码长度 3 者之间的关系,适用于软件开发的各个阶段。

Putnam 模型把项目的资源需求当作时间的函数,根据对一些大型项目的统计分析,软件开发工作量分布可用如图 4-4 所示的曲线表示。图中的工作量分布曲线的形状与著名的 Rayleigh-Norden 曲线相似。

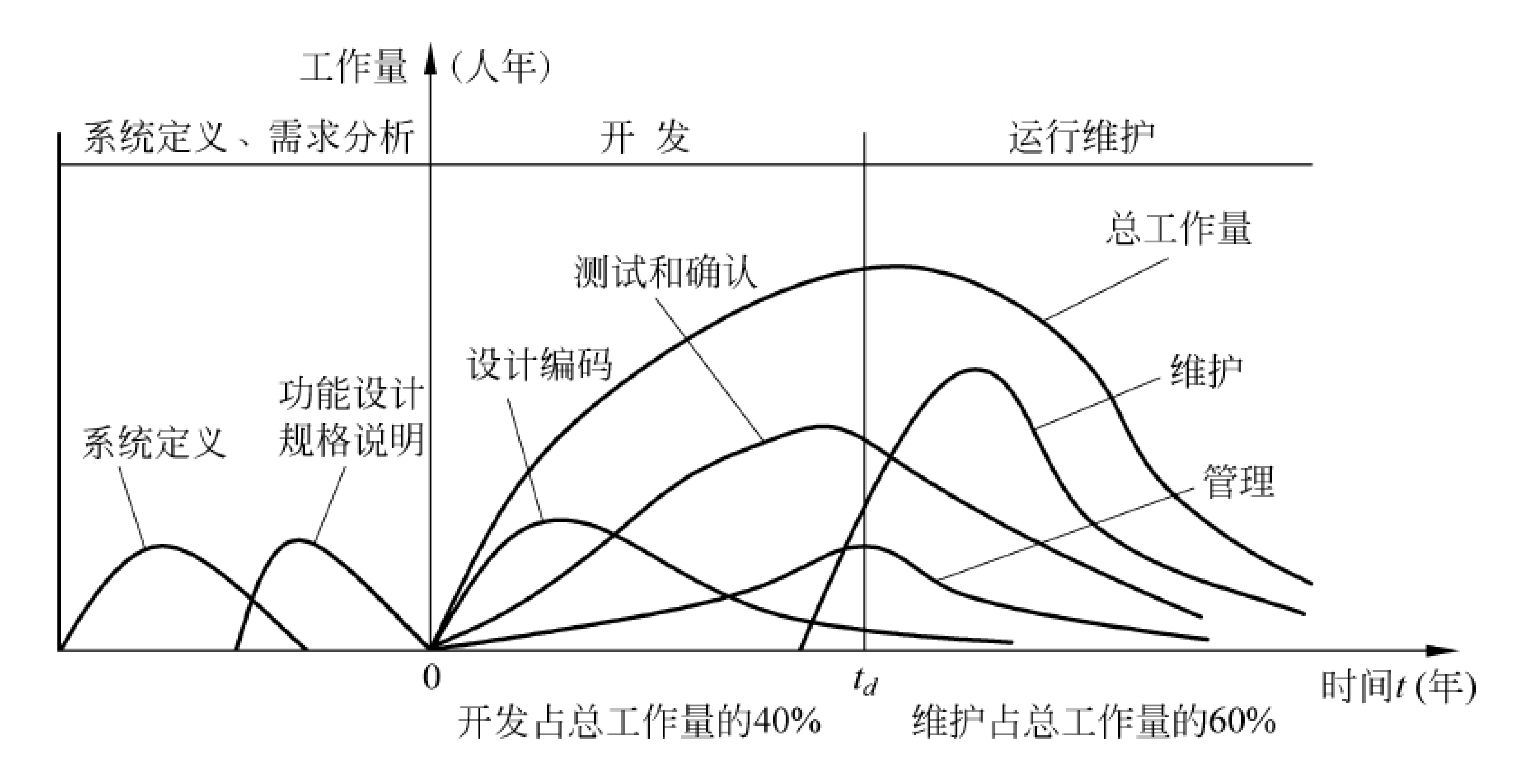


图 4-4 软件项目的工作量分布曲线

由图 4-4 可得出 Putnam 估算模型如下:

$$L = P \times E^{1/3} t_d^{4/3} \tag{4-13}$$

其中:L 为源代码行数(以 LOC 计);E 为开发与维护的工作量(以人年计);t_d 为开发时间(以年计);P 为生产率参数,与开发环境有关,取值如表 4-10 所示。

表 4-10 不同开发环境的生产率参数

开发环境	描 述	P 值
差的软件开发环境	软件开发没有方法学的支持,缺乏对文档的评审,采用批处理方式	2000
一般的软件开发环境	应有软件开发方法学的支持,有适宜的文档和评审,采用交互处理方式	8000
好的软件开发环境	应采用 CASE 工具和集成化 CASE 环境	11 000

由 Putnam 基本估算公式,可得出工作量及开发时间之间的关系:

$$E = L^3/(P^3 t_d^4) \tag{4-14}$$

或

$$t_d = \sqrt[4]{L^3/(P^3 \cdot E)}$$

从式(4-14)中可知,软件开发项目的工作量(E)与交付时间(t_d)的4次方成反比,显然,软件开发过程中人员与时间的折中是十分重要的问题。Putnam 将这一结论称为"软件开发的权衡定律"。

此外,Brooks 从另一个角度说明了"时间与人员不能线性互换"这一原则。

对上述定律的合理解释是,当开发人员以算术级数增长时,人员之间的通信将以几何级数增长,从而可能导致"得不偿失"的结果。一般说来,由 N 个开发人员组成的小组要完成既定的工作,相互之间的通信路径总数为 $C_N^2=N(N-1)/2$,而通信是需要时间的。所以,当新的开发人员加入项目组之后,原有的开发人员必须向新来的成员详细讲解某个活动或工作包的来龙去脉。由于信息系统开发具有较强的个人风格,所以交流沟通的时间更容易拉长,而后来者还不一定能达到原来开发人员的工作质量。

【例 4-4】 根据 Putnam 模型,Et_d^4=常数,其中 E 为工作量(人年),t_d 为开发时间(年)。如果将开发时间减少到原计划的50%,其工作量将增加多少倍?

解:根据 Putnam 估算模型 $L=P\times E^{1/3}t_d^{4/3}$ 可得:

$$E = L^3/(P^3t_d^4)$$

于是工作量增加的倍数 K 为:

$$K=\frac{L^3/(P^3(0.5t_d)^4)}{L^3/(P^3t_d^4)}=16$$

即工作量将增加16倍。

4.4　软件成本估算

计算机软件是信息社会的重要商品,也是知识经济社会中的重要资产。软件项目管理是为了使软件项目能够按照预定的成本、进度、质量顺利完成。所以,软件成本管理是软件项目管理的一个主要内容。

要做出一个明智的决定,必须懂得如何正确地计算成本。经济学中所讲的成本,指的是将来经济活动的开展所消耗的物品和资源,这些统统称做机会成本。对于已经花费的、不可能再收回的费用,或者说不管采取什么手段都无法再节省的费用,经济学称之为沉没成本。这些原则同样适用于软件成本。

当一个项目按合同进行时,应区分软件成本估算和软件定价这两个不同意义的词。成本估算涉及的是对可能数量结果的估算,即软件组织为提供产品或服务的花费是多少。而定价是一个商业决策,即组织为它提供的产品或服务索取多少费用,而成本估算只是定价要考虑的因素之一。

4.4.1　软件项目成本的组成

软件项目成本是指在其生命周期(系统规划、分析、设计、构建与运行维护阶段)内,为取得各种软硬件资源的支持及维持系统的研究、生产经营与管理正常开展所投入的人、财、物

而支付的一切费用①。

软件项目成本是软件定价的基础。从财务角度来看，列入软件的成本有如下项目：①硬件购置费，如计算机及相关设备的购置；②软件购置费，几乎所有的软件都需要有其他软件的支持才能运行，除了系统本身提供的软件外，一些具有特定功能的软件需要从第三方购买，如数据库系统软件、第三方软件、可复用的构件、测试软件等费用；③人工费，主要是开发人员、操作人员、管理人员的工资等费用；④培训费；⑤通信费，如购置计算机网络设备、通信线路器材、租用公用通信线路等的费用；⑥基本建设费，如新建、扩建机房、购置计算机机台、机柜等的费用；⑦财务费用；⑧管理费用，如办公费、差旅费、会议费、交通费；⑨材料及耗材费用；⑩水、电、运输费；⑪专有技术购置费；⑫其他费用，如资料费、固定资产折旧费及咨询费。

从软件生命周期构成的两阶段即开发阶段和维护阶段看，系统软件的成本由开发成本和维护成本构成，如图 4-5 所示。其中开发成本由软件开发成本、硬件成本和其他成本组成，包括了系统软件的分析/设计费用（含系统调研、需求分析、系统分析）、实施费用（含编程/测试、硬件购买与安装、系统软件购置、数据收集、人员培训）及系统切换等方面的费用；维护成本由运行费用（含人工费、材料费、固定资产折旧费、专有技术及技术资料购置费）、管理费用（含审计费、系统服务费、行政管理费）及维护费用（含纠错性维护费用及适应性维护费用）。

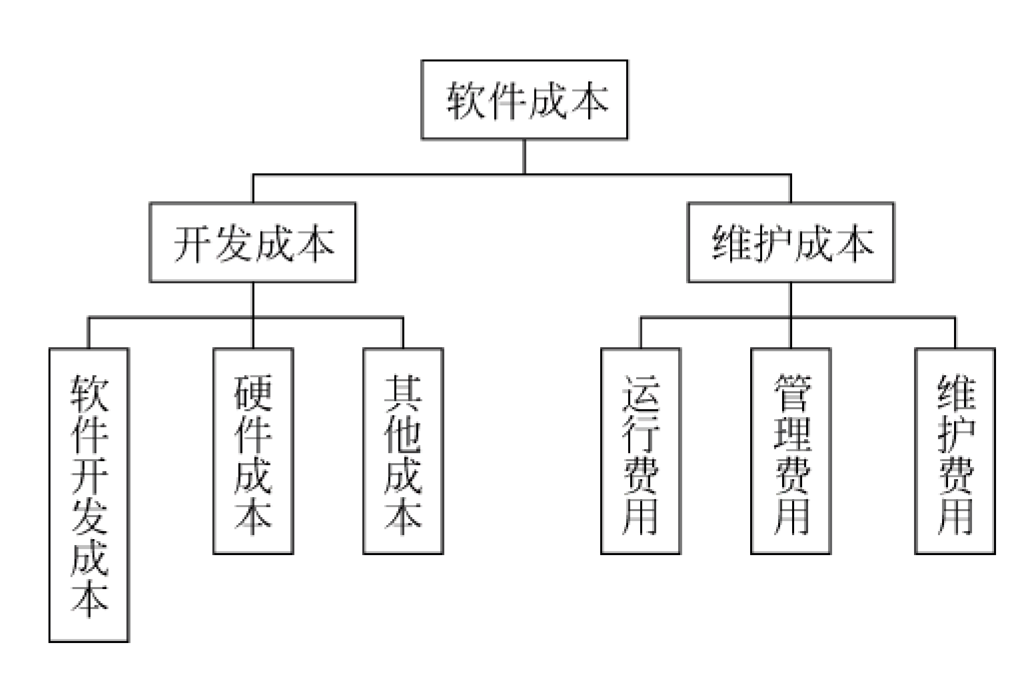


图 4-5 软件成本的构成

4.4.2 软件成本的估算方法

软件成本估算在软件工程经济学中有着举足轻重的分量，因为在软件工程经济学的分析中所用到的现金流量，大部分都是基于软件成本估算而获得的。目前，已有较多软件成本估算方法可供使用，其中较为常见的有自顶向下估算法、自底向上估算法、参数模型估算法、专家估算法、表格法等。

1. 自顶向下与自底向上估算法

在项目初期，可以采用某种估算公式估算出系统总的成本。一旦我们获知了项目的总体工作量，将开发的软件项目分解为若干个相对独立的任务，可将软件项目分解为若干个子系统，再将子系统按开发阶段划分成更小的任务，分别估计单独开发每个任务的成本，最后累加起来得到软件开发项目的总成本。这种估算法的优点是估算的工作量小，速度快；缺点是对项目中的特殊困难估计不足，估算出来的工作量盲目性大，有时会遗漏某些部分。

自底向上估算法通常先估计完成项目中每项任务所需的人力时间（以人月为单位），再

① 赵玮. 软件工程经济学. 陕西：西安电子科技大学出版社，2008

乘以每人每月的平均工资而得出每个任务的成本，最终得到整个系统的成本。这种估算法的优点是估算各个部分的准确性高；缺点是缺少各项子任务之间相互联系所需要的工作量，还可能缺少许多与软件开发有关的系统级工作量（配置管理、质量管理、项目管理），所以往往估算值偏低。

2. 专家评估法——Delphi 方法

Delphi 法是最流行的专家评估技术，适用于规模、工作量、工期、成本等各种估算，美国兰德（Land）公司首先于 1964 年把 Delphi 法用于技术预测中。Delphi 是在专家个人判断和专家会议方法的基础上发展起来的一种直观预测方法，特别适用于客观资料或数据缺乏情况下的长期预测，或其他方法难以进行的技术预测。

Delphi 法是以专家作为索取信息的对象，依靠专家的知识和经验，由专家通过调查研究对问题做出判断、评估和预测的一种方法。专家通过自己的经验和对所提及项目的理解，得出该项目的成本估算值，其大致流程如下。

（1）组织者发给每位专家一份软件系统的规格说明书（略去名称和单位）和一张记录估算值的表格，请他们进行估算。

（2）专家无记名填写表格，专家不得互相讨论，但可以向组织者提问。

（3）组织者对结果进行总结并把结果发给专家，专家进行估计值调整，并说明理由，如图 4-6 所示。

（4）专家再次无记名填写表格。

（5）根据估计情况重复以上 4 个步骤直到估算结果偏差可以接受为止。

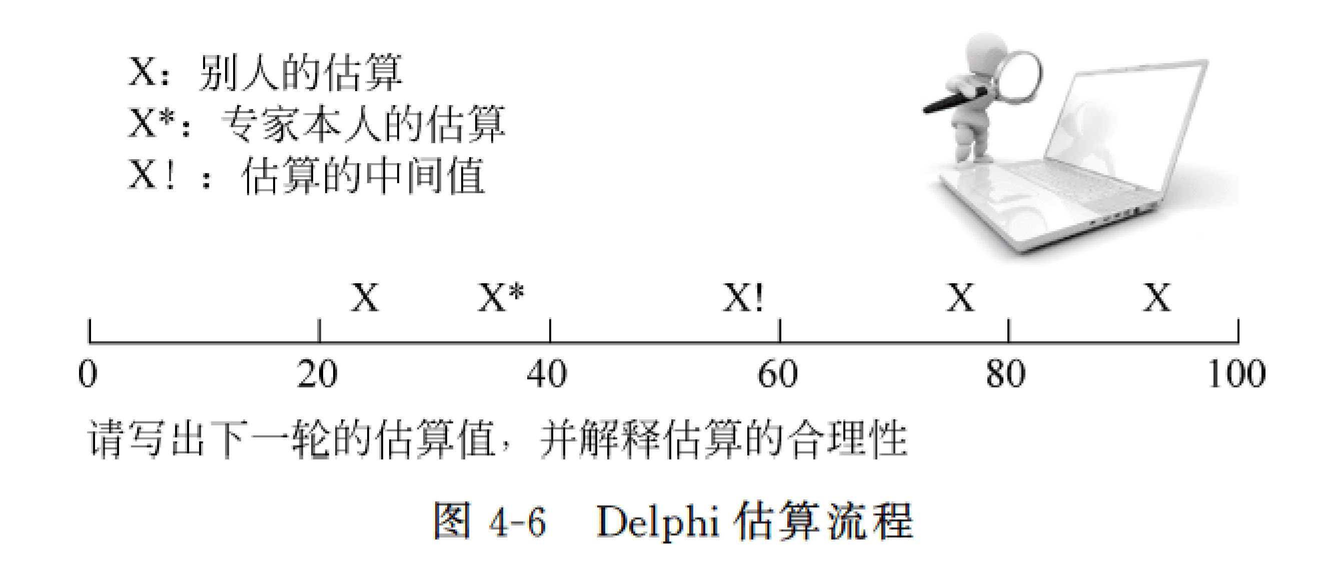


图 4-6　Delphi 估算流程

由此可见，Delphi 法是一种利用匿名形式的集体思想交流过程。它区别于其他专家预测方法的 3 个明显的特点，分别是匿名性、多次反馈、小组的统计回答。

3. 基于模型的估算方法

基于模型的估算法是一种使用项目特征参数建立数学模型来估算成本的方法。估算者首先要有一个模型。对模型的基本要求是它能够描述软件成本和成本驱动因子间的关系，并用这一模型对样本数据进行拟合，得出其参数值。其中的项目特征参数构成了成本驱动因子，最终估算得到的成本是这些成本驱动因子的函数。用于软件成本估算的主要参数模型形式有：静态单变量模型，静态多变量模型，动态多变量模型。

（1）静态单变量模型

静态单变量模型将待估算的各种资源都同某个单一的已估算的特性联系起来，这当然

是一种简化。这种模型的形式是：

$$资源 = a \times (已估算的特性)^b$$

其中：资源可以是工作量(E)、软件规模或项目周期(D)；“已估算的特性”可以取为源代码行数，工作量或其他的软件特性；两个常数 a 和 b，则需要根据以前项目中收集的数据计算得出。

前面给出的 Walston-Felix 模型和 Basili 模型都属于这一类。

(2) 静态多变量模型

在这种模型里，所用的公式形式同单变量模型公式相似，不同的是待估成本依赖于不同的已估算的特性，COCOMO 模型即属于这一类。

(3) 动态多变量模型

动态多变量模型把项目资源需求作为时间的函数。如果模型是依据经验得出的，则这些函数表示在一系列时间段上定义的资源。建立动态多变量模型的一个理论方法是假定一条连续的“资源使用曲线”，并依据这条曲线得出模型化资源特性的公式，Putnam 模型就是这样一个模型。

4. 表格模型

表格法的基本思想是将与软件成本有关的工程经济参数，如系统规模、复杂性、工期(进度)、系统需求及约束、劳动生产率、工时费用率以及对成本有影响的其他各类各种影响因子有机地组织起来，并汇总成几张表格，然后系统设计人员可根据软件的功能需求及约束条件，按照表格填写的顺序进行填写与计算，并最终完成对软件的成本与工期参数的估算。例如，功能点分析、详细 COCOMO 模型等都可视为利用表格法做成本估算的应用案例。

5. 类比估算法

类比推理是人们解决问题过程中的一个基本的策略。类比估算又被称为基于案例的推理，适合评估一些与历史项目在应用领域、环境和复杂度等方面相似的项目。类比估算法是通过将新项目的估算成本与一个或多个已完成项目的实际成本联系起来，用类推的方法来推算新项目的成本。通过类比来估算，可以在整个项目的级别上进行，也可以在子系统级别进行。整个项目级别的估算有这样的优点：所有的系统成本组件都会被考虑到。子系统级别的估算有这样的优点：它对新项目与已完成项目之间的异同能提供更详细的评价。

类比估算法最主要的优点在于该估算是基于实际的项目经验。可以通过研究以往经验来确定与新项目之间的具体差异，及其可能对成本产生的影响。其主要不足是，以前项目能在多大程度上代表新项目的约束条件、技术、人员与软件执行的功能，还需要仔细权衡。另外，使用该方法对项目进行估算时，要保证收集的数据是正确的，否则将没有参考价值。如果当需要估算的项目与历史项目之间存在明显差异时，不宜使用该方法。

6. 神经网络方法与模糊理论方法

人工神经网络是采用工程技术手段模拟人脑神经网络的结构和功能的一种技术系统。

它是由大量的、同时也是很简单的处理单元即神经元,通过广泛地相互连接而形成的复杂网络系统。虽然每个神经元的结构和功能十分简单,但它们构成了一个高度复杂的非线性动力学系统,具有高维性、神经元之间的广泛相连性以及自适应性等特点。

由于人工智能控制系统有着极强的信息处理能力,有学者逐渐将智能控制理论应用到软件成本估算中。例如,有研究者将神经网络和集合分析理论相结合用于软件开发成本估算,提高估算的效率;有学者用模糊逻辑方法对COCOMO模型中的软件成本驱动因子进行分类模糊处理,来解决不确定信息和主观判断对成本估算造成的影响。这些研究都有一定的成效,但由于神经网络和模糊逻辑各自的局限性,任何一种单纯的方法都不能较完全地解决软件成本估算中的问题。

模糊理论是在模糊集合理论的数学基础上发展起来的,主要包括模糊集合理论、模糊逻辑、模糊推理和模糊控制等方面的内容。模糊逻辑在处理不确定信息方面有重要的作用。模糊理论中的隶属度函数可以表达一个模糊概念从"完全不属于"到"完全隶属于"的过渡,对于一些常规经典集合中无法准确表示的模糊概念可以有效地表达。模糊推理可以在所获得的模糊信息的前提下进行有效的判断和决策。综合应用了模糊理论各要素的模糊控制系统能够解决传统控制理论无法解决或难以解决的问题。

4.4.3　估算技术的应用与评价

软件工程成本估算的基本目标是为软件人员在资源有限的条件下如何做出决策提供更好的定量的理解。对于有兴趣使用估算技术的软件人员来说,最为关键的问题是,所提供的估算是否足以证明使用它们的成本(在某些情况下购买)是正当合理的。在理想的情况下,一个模型应该易于使用,并提供足够精确的估算,而无须额外的评估费用。

国外的一些研究人员对不同软件项目估算技术在不同国家软件开发组织中的应用情况进行了调查,Kjetil Molkken等对1984～2002年期间进行的关于软件项目工作量估算的各种调查进行了概括总结,他们把软件项目估算技术分为3类①:基于专家的估算方法、基于模型的估算方法和其他方法。研究指出,由于不同调查所采用的估算技术分类及被调查人对分类的解释不尽相同,要严格地比较这些调查结果是困难的。不过从这些调查还是看到了一个明显的共同之处,即基于专家判断的估算技术是至今为止使用最普遍的软件项目估算技术。这可能是因其所具有的易用性和灵活性,在缺少量化的、历史数据的情况下该技术非常有用。但是这种方法依赖专家的经验、背景和成熟度来猜测一项任务或整个项目需要多长时间完成或成本多少,因此,专家"专"的程度及对项目的理解程度是估算工作中的难点,也是这种估算技术的局限性。尤其当进行新产品开发,企业不具有开发类似项目的经验时,该方法很难实施。另外,该技术如何导出估算值的方法不清晰,是不可重复的,因此,软件开发组织很难清晰定义这种估算过程,不能获得组织学习的效应。

基于算法模型的估算技术与基于专家判断的估算技术相比,其主要优点是让用户看到了一个模型如何计算并得出结论的具体过程。虽然文献关于这一类估算技术的研究很多,

① 王求真.软件开发项目工作量估算技术的比较研究[J].浙江大学学报,2005,35(4)

但该技术在实际应用中还未得到广泛应用，一方面是因为需要对估算模型理解透彻，另外也缺乏证据表明这类方法比基于专家的估算技术估算更准确。一些研究人员对不同估算模型的精确度进行验证，从各个方面来看，没有哪一个可选方法比所有其他的方法强。其余方法的优点和缺点是互补的(特别是算法模型与专家评价法、自上而下法与自下而上法)。因此，在实际应用中，应该综合应用以上的各种方法，反复比较它们的结果，寻找这些结果到底在哪些地方不同。

越是大型的软件项目，项目成本估算的工作量越大，要求进行估算的时间却往往很早，甚至在系统需求尚未明确、项目任务还没有被详细确定之前。因为选择项目、提出商业报价、估计利润等都需要一个成本的基本估计。所以，实际存在着不同级别的成本估计——系统级、模块级和 WBS 级。项目在不同阶段和级别上的估计是同样重要的，项目管理者必须能够解释在不同级别上估计的一致性和合理性。否则，就失去了估算的意义。

在软件工程领域，已经开展了许多经验性研究对各种成本估算模型进行验证。验证对于确定模型是否有价值是非常重要的。然而，对于软件项目，大多数人没有太多的成本估算的经验和历史数据，特别是对大型项目而言更是如此。由于需要获取大量的已完成项目的数据，所以模型验证非常困难。尤其是对于大型软件项目，这些数据的收集很难获得。随着计算机辅助软件工程(CASE)工具的出现，新模型的开发将会取得不断的进展。这些为系统开发者设计的工具将极大地帮助成本估算研究人员收集诸如在软件生命周期各阶段(如分析、设计、构建、测试阶段等)的数据。这种数据收集的自动化，将减少生成建立在分析和设计度量基础上的新模型的障碍。

【例 4-5】 综合估算案例：基于代码行的 CAD 软件包估算

某软件组织准备开发一个小型 CAD 软件包，并采用自底向上的估算方法，对软件项目的成本和工作量进行了初步估算，其估算步骤如下。

(1) 列出开发成本表。

首先，将 CAD 项目按功能分解为 7 个子项目，并估算出每个子项目 LOC 的乐观值 a、一般值 m 和悲观值 b(如表 4-11 所示)。考察前 3 列数据估算的范围，估算者对外设控制功能模块的规模是较有把握的，因为乐观的和悲观的估算值只差 450LOC。另外，对于三维几何图形分析这个功能块，是心中无数的，差了 4000LOC。

表 4-11 CAD 软件的估算值

子项目	a LOC	m LOC	b LOC	期望值 X LOC	标准差 S	每行成本 元/LOC	生产率 P LOC/PM	成本 C 元	工作量 PM
用户接口控制	1800	2400	2650	2340	140	14	315	32 760	7.4
二维几何造型	4100	5200	7400	5380	550	20	220	107 600	24.4
三维几何造型	4800	6900	8600	6800	670	20	220	13 600	30.9
数据库管理	2900	3500	3600	3350	110	18	240	60 300	13.9
图形显示	4000	4900	6200	4950	360	22	200	108 900	24.7
外设控制	2000	2100	2450	2140	75	28	140	59 920	15.2
设计分析	6600	8500	9800	8400	540	18	300	151 200	28.0
总计				33 360	1100			656 680	145

（2）求期望值和标准差。

按照三点估算公式和标准差计算公式，计算出每个子项目代码行的期望值 X 和标准差 S，计算结果列于表中的第 4、5 列。整个 CAD 软件的源代码行数的期望值为 33 360，标准差为 1100。根据标准差的含义，可以假设 CAD 软件的规模在 32 000～34 500LOC 的概率为 0.68，软件规模在 30 000～36 700LOC 的概率在 0.99。可以应用这些数据得到软件的成本和工作量的变化范围，或者用于项目的风险分析。

（3）根据已知的开发类似子项目的生产率 P，以及每代码行平均成本 C，即可估算出每一个子项目的成本和工作量，最后将 7 个子项目的成本和工作量分别累加，即可估算出软件项目的总成本 C 和总工作量 E，分别为 657 000 元和 145 人月。

（4）使用表中的有关数据求出开发时间。

① 用 Putnam 方程计算开发时间

假设此软件处于"正常"开发环境，即 $P=10\,000$，并将 $L=33\,000$，$E=145$ 人月 $=12$ 人年，代入 Putnam 方程：

$$t_d = \sqrt[4]{L^3/(P^3 \cdot E)}$$

则需要的开发时间为：

$$t_d = \sqrt[4]{33\,000^3/(10\,000^3 \times 12)} = 1.3(\text{人年})$$

② 用基本 COCOMO 模型计算开发时间

根据基本 COCOMO 模型：

$$D = cE^d$$

其中：E 是以人月为单位的工作量；D 是以月为单位的开发时间；假设此软件为"组织模式"，式中的参数 c，d 按表 4-7 取值。将 $c=2.5$，$d=0.38$ 代入 COCOMO 基本公式：

$$D = 2.5 \times 145^{0.38} = 16.57(\text{人月}) = 1.38(\text{人年})$$

由此看出，用两种估算方法得出的开发时间基本上是一致的。

正态分布的规则

在实际应用中，考虑一组数据具有近似于正态分布的机率分布，若其假设正确，则约 68%数值分布在距离平均值有 1 个标准差之内的范围；约 95%数值分布在距离平均值有两个标准差之内的范围；以及约 99.7%数值分布在距离平均值有 3 个标准差之内的范围。这称为"68-95-99.7 规则"。

【例 4-6】 用任务分解技术估算 CAD 软件成本与工作量

对于项目成本估算，可以采用任务分解技术，将每个子项目按生存周期划分，估算其各阶段的工作量，再累加求出每个子项目的工作量和整个项目的工作量，并将估算的结果与模型法进行比较，来验证估算的准确性。

解：（1）任务分解

确定每个子项目（任务），每个子项目（功能）都必须经过需求分析、设计、编码和测试阶段。

（2）确定每个子项目的工作量

对每个子项目按阶段估算它们所需要的人月数（表 4-12）。

表 4-12 按任务分解技术

子项目	需求分析	设计	编码	测试	总计
用户接口控制	1.0	2.0	0.5	3.5	7.0
二维几何造型	2.0	10.0	4.5	9.5	26.0
三维几何造型	2.5	12.0	6.0	11.0	31.5
数据库管理	2.0	6.0	3.0	4.0	15.0
图形显示	1.5	11.0	4.0	10.0	27.0
外设控制	1.5	6.0	3.5	5.0	16.0
设计分析	4.0	14.0	5.0	7.0	30.0
总计	14.5	61.0	26.5	50.5	152.0
人员费用(元/人月)	5200	4800	4250	4500	
成本	75 400	292 800	112 625	227 250	708 000

(3) 列出与各项任务对应的人员费用

即每个单位工作量成本(元/人月)。因为各阶段的人员费用不同,需求分析和概要设计阶段需要较多的高级技术人员;而详细设计、编码和早期测试则要求较多的初级技术人员。而向这些员工所支付的工资等费用是不同的。

(4) 计算

计算各个工作各个阶段的成本和工作量,然后计算总成本和总工作量。

(5) 分析比较

在整个开发工作量中,需求分析和设计用去了 75 人月,约占全部任务工作量的 50%,说明了这项工作的重要性。人员费用反映了劳动者的成本,其中包括管理费。需求分析的人员费用(5200 元/人月)比设计、编码和测试都高,这也说明了这项工作的重要性。

用这种方法估算,CAD 软件成本和工作量总计为 708 000 元和 152 人月。将这些数据与代码行的成本估算比较(分别为 656 680 元和 145 人月),前者相差 7%,后者相差 5%,结果非常接近,可以接受。若相差较大,应该分析原因,很多情况下是由于估算人员对任务没有完全理解或有误解,或使用的员工费用数据不够恰当。估算者必须找出原因后再重新估算,结果应基本一致。

4.5 项目进度估算

即使规模估算很好,生产率数据也很可靠,但进度仍然是一个很大的未知数。进度估算过程要求对整个作业所需的时间进行详细的估算;然后再把这些时间分配到各个项目阶段中。

4.5.1 三点估算方法

三点估算方法不仅可以用于成本估算,同样可用于项目进度估算。项目进度是个随机变量。如果可能,活动在给定条件下重复进行几次,工期也会有差别。变量也许紧密围绕着一个中心值,也许非常离散。对于第一种情况,可能有大量有关活动工期的有用信息。相比之下,后一种情况,可能只有很少信息,甚至没有。对于任何活动情况,很难估算出准确的活

动工期,但我们总可以对发生的可能性进行判断。

三点估算技术给我们的判断提供了一个框架。这个方法要求估算者对项目活动的完成时间做3类估计:乐观的、悲观的和最可能的。

(1) 乐观时间 a_i:事情都顺利的情况,完成某项工作的时间。

(2) 最可能时间 m_i:正常情况下,完成某项工作的时间。

(3) 悲观时间 b_i:最不利的情况,完成某项工作的时间。

假定3个估计服从 β 分布,由此可算出每个活动的期望 E_i 和总的时间期望 E。

$$E_i = \frac{a_i + 4m_i + b_i}{6}, \quad E = \sum_{i=1}^{n} E_i$$

根据 β 分布的方差计算方法,第 i 项活动的持续时间方差和标准差为:

$$\sigma_i^2 = \left(\frac{b_i - a_i}{6}\right)^2, \quad S = \sqrt{\sum_{i=1}^{n} \sigma_i^2}$$

【例 4-7】 假如一个新的通信程序,活动分别为需求、设计和编码测试,各活动估计的悲观、期望、乐观时间分别为:需求(7,11,15)天,设计(14,20,32)天,编码(20,30,40)天,试计算整个项目完成时间的数学期望和标准差。

解:

$$E = \sum_{i=1}^{n} E_i = [(7 + 11 \times 4 + 15) + (14 + 20 \times 4 + 32) + (20 + 30 \times 4 + 40)]/6 = 62$$

$$S = \sqrt{\sum \sigma_i^2} = \sqrt{[(15-7)^2 + (32-14)^2 + (40-20)^2]/36} = 3.83$$

据此,根据正态分布规律,在 $\pm\sigma$ 范围内即在65.83(62+3.83)天与58.17(62−3.83)天之间完成的概率为68%。经验表明,估计偏低的倾向大于偏高的倾向,决策时需注意。

4.5.2 项目进度获取值分析——项目计划与实际进展的定量比较

进度获取值分析(Earned Value)是估计项目进展的一种方法,为所有任务提供了一种通用的价值尺度,而不管所涉及工作的类型。这种方法为每个任务建立了一个相对价值,并且在任务完成时记录了成果。

如果所规划的项目有多个任务,而且要以一种特定的次序来完成这些任务,这时,尽管可能会修改这种任务次序,我们还是希望有一种方法来判断进展。如果所有的任务所占用的时间都一样,或者说只有很少的任务所占用的时间不一样,这时还不算很困难。但如果很多任务的规模都不相同,这时就变得很复杂了。

获取值分析有以下3个基本度量。

(1) 计划完成工作的预算成本(Budgeted Cost of Work Scheduled,BCWS):是到目前为止的总预算成本,它表示“到目前为止原来计划成本是多少”或者说“到该日期为止本应该完成的工作是多少”,它是根据项目计划计算出来的。

(2) 已完成工作的实际成本(Actual Cost of Work Performed,ACWP):是到目前为止所完成工作的实际成本,它说明了“到该日期为止实际花了多少钱”,可以由项目组统计。

(3) 已完成工作的预算成本(Budgeted Cost of Work Performed,BCWP):又称为已获取价值,是到目前为止已经完成的工作的原来预算成本,它表示了“到该日期为止完成了多

少工作”。

理想状态下 BCWP、BCWS、ACWP 这 3 条曲线可以重合，但大多数情况下很难一致，如图 4-7 所示。

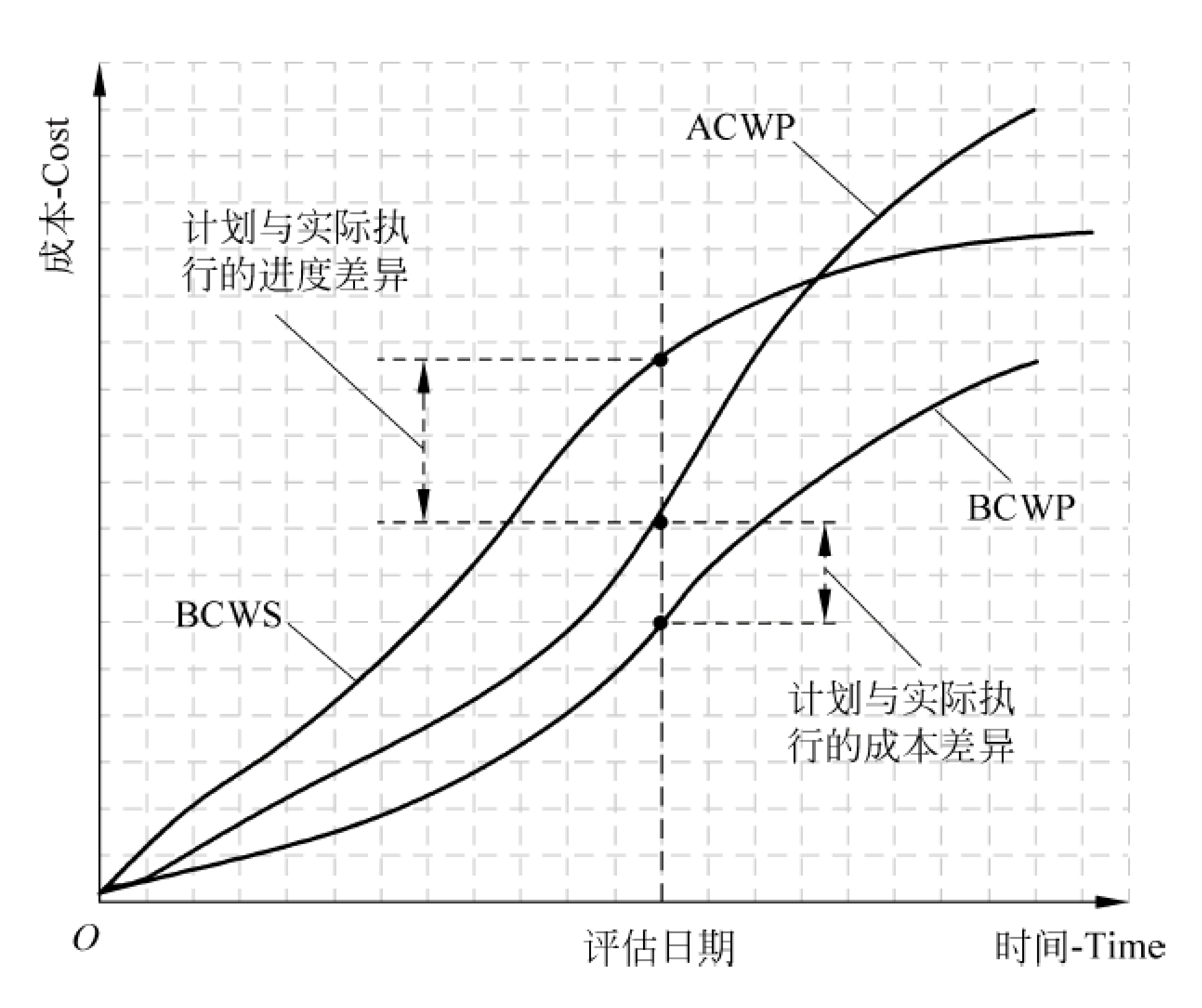


图 4-7 获取值分析的 3 个基本度量

在某一分析日期得到项目的 3 个基本度量之后，就可以确定到目前为止项目的进度和预算情况。进度和预算情况通过下面 4 个导出度量来获得，其分析过程如下。

(1) 进度差异：SV(Schedule Variance)＝BCWP－BCWS。若此值为零，表示按照进度进行；如果为负值，表示项目进度落后；如果为正值，表示进度超前。

(2) 费用差异：CV(Cost Variance)＝BCWP－ACWP。若此值为零，表示按照预算进行；如果为负值，表示项目超出预算；如果为正值，表示低于预算。

(3) 进度效能指标：SPI(Schedule Performance Index)＝BCWP/BCWS。表示完成任务的百分比。若此值为 1，表示按照进度进行；如果小于 1，表示项目进度落后；如果大于 1，表示超进度进行。

(4) 成本效能指标：CPI(Cost Performance Index)＝BCWP/ACWP。表示花钱的速度。若此值为 1，表示按照预算进行；如果小于 1，表示项目超出预算；如果大于 1，表示低于预算。研究表明：项目进展到 20%左右，CPI 应该趋于稳定，如果这时 CPI 值不理想，应该采取措施，否则这个值会一直持续下去。

明确了项目当前的进度和预算情况后，可以在此基础上对项目进行预测。预测中用到几个导出度量，下面给予定义。

(1) 工作完成的预算成本(Budgeted At Completion，BAC)：是项目计划中的成本估算结果，是项目完成的预计总成本。

(2) 项目完成的预测成本：EAC(Estimate At Completion)＝BAC/CPI。

(3) 项目完成的成本差异：VAC(Variance At Completion)＝BAC－EAC。

(4) 项目完成的预测时间：SAC(Schedule At Completion)＝完成时的进度计划/SPI。

4.6 软件工程经济学

软件工程研究不仅限于软件开发技术和软件工程管理两个方面，还取决于它是否满足经济学和社会效益的需要。纵观软件工程开发的全过程，从经济学的角度来看，软件项目管理和软件成本估算都与工程经济密切相关。软件工程经济学就是依据经济学的观点来研究如何有效地分析、开发、发布软件产品及其支持用户使用等，为软件的成本、进度估算提供必要的手段和方法，并妥善地协调技术、经济和人之间的关系。

4.6.1 经济学与工程经济学

经济学是对人类各种经济活动和各种经济关系进行理论的、应用的、历史的以及有关方法的研究的各类学科的总称。经济学又可称为经济科学(Economic Sciences)，是经世济民的科学，是研究人类个体及其社会在自己发展的各个阶段上的各种需求和满足需求的活动及其规律的学科。按照研究的范畴不同，经济学可划分为宏观经济学(Macro Economics)和微观经济学(Micro Economics)。**宏观经济学**是以国民经济总过程的活动为研究对象，因为主要考察就业总水平、国民总收入等经济总量，因此，宏观经济学也被称为就业理论或收入理论。微观经济学主要以单个经济单位(单个的生产者、单个的消费者、单个市场的经济活动)作为研究对象，分析单个生产者如何将有限的资源分配在各种商品的生产上以取得最大的利润；单个消费者如何将有限的收入分配在各种商品的消费上以获得最大的满足。

按照所研究领域的不同，经济学又可有工程经济学、管理济学、发展经济学、信息经济学、经济统计学等分支学科。工程经济学(Engineering Economics)是工程与经济的交叉学科，是研究工程技术实践活动经济效果的学科。即以工程项目为主体，以技术-经济系统为核心，研究如何有效利用资源，提高经济效益的学科。工程经济学研究各种工程技术方案的经济效益，研究各种技术在使用过程中如何以最小的投入获得预期产出或者说如何以等量的投入获得最大产出；如何用最低的寿命周期成本实现产品、作业以及服务的必要功能。

4.6.2 软件工程经济学研究的基本问题

如果我们着眼于软件工程这门学科，会看到作为经济学分支学科的微观经济学，为软件工程师或者软件管理者必须做出的决策提供了相应的支持。很明显，我们面对的是有限的资源。我们永远不会有足够的时间或者金钱去把我们想要的所有好的特性加入到软件产品中。在整个软件生命周期内，有许多包括有限资源在内的决策状况，而软件工程经济学能为此提供有用的帮助。

Boehm 是第一个从经济学角度看待软件工程学的研究者，他的经典著作《软件工程经济学》奠定了软件成本估算领域的基础。软件工程经济学(Software Engineering Economics)是软件工程学科与经济学(主要是微观经济学)有机结合的产物，它利用经济学中成熟的概念、技术和方法为软件工程决策服务(图 4-8)。因此，软件工程经济学面临的问题是如何利用成本估算等技术来帮助项目管理者做出正确的选择，以及利用何种经济学方

法帮助人们做出正确的决策。



图 4-8 软件工程经济学是软件工程学科与经济学的交叉学科

作为一门有待发展的新兴学科，从软件项目管理的视角来分析，软件工程经济学研究的基本问题如下。

（1）成本、工作量、生产率等因素的估算技术与方法以及估算模型的建立与使用，这是一个核心问题。

（2）软件工程中不同决策的“成本效益”分析，及其与此相关的规模经济与不经济问题。

（3）多目标决策分析，以便识别目标、协调与决策相互冲突的目标、管理多个同时存在的目标等。

（4）成本、工作量、人力分布及其资源配置问题。

（5）不确定性的处理和风险分析问题。

（6）进度估计和工期控制问题。

（7）数据收集与管理以及模型的校准等问题。

（8）相关工具问题。

4.6.3 资金的时间价值

1. 资金时间价值的概念

资金的时间价值是指资金在运动过程中，随时间的推移而发生的增值。一笔用于项目投资的资金，投入后经过一定时间，由于净效益的产生，使原资金得到增值，即获得了较原投资额更多的资金。资金时间价值揭示了不同时点上一定数量的资金之间的换算关系，它是进行投资、筹资决策的基础依据。

如果将资金锁在柜子里，无论如何也不会增值。人们将资金在使用过程随时间的推移而发生增值的现象，称为资金具有时间价值的属性。资金时间价值是资金在周转使用中产生的，是资金所有者让渡资金使用权而参与社会财富分配的一种形式。例如，将今天的 100 元钱存入银行，在年利率为 10%的情况下，一年后就会产生 110 元，可见经过一年时间，这 100 元钱发生了 10 元的增值。

2. 时间价值的计算

由于资金具有时间价值，因此同一笔资金，在不同的时间，其价值是不同的。计算资金的时间价值，其实质就是不同时点上资金价值的换算。计算货币时间价值量，首先引入“现值”和“终值”两个概念表示不同时期的货币时间价值。

现值(Present Value),又称本金,是指资金现在的价值。计算未来时点上一定数额的资金,相当于现在多少数额的资金,这是计算现值问题。

终值(Future Value),又称本利(本金与利息)和,是指资金经过若干时期后包括本金和时间价值在内的未来价值。现在拥有一定数额的资金,在未来某个时点将是多少数额,这是计算终值。

资金时间价值的计算方法和有关利息的计算方法类似,因此资金时间价值的计算涉及利息计算方式的选择。有单利计息和复利计息两种方法。单利是指仅用本金计算利息,利息不再生利息。复利是指用本金与前期累计利息总额之和计算利息,也就是除最初的本金要计算利息之外,每一计息周期的利息也要并入本金再生利息。在项目经济分析中,一般均采用复利计息。

资金时间价值的计算通常有单利终值与现值、复利终值与现值、年金终值与现值 3 种形式。

(1) 单利终值与现值

单利是指只对借贷的原始金额或本金支付(收取)的利息。我国银行一般是按照单利计算利息的。

在单利计算公式中,设定以下符号:

P:现值;i:利率;F:终值;t:时间。

① 单利终值计算

单利终值是本金与未来利息之和,其计算公式为:

$$F = P(1 + i \times t) \tag{4-15}$$

【例 4-8】 将 100 元存入银行,利率假设为 10%,一年后、两年后、三年后的终值是多少(单利计算)?

解:按照单利终值公式有:

一年后:$F=100\times(1+10\%)=110$(元)

两年后:$F=100\times(1+10\%\times2)=120$(元)

三年后:$F=100\times(1+10\%\times3)=130$(元)

② 单利现值计算

单利现值是资金现在的价值。单利现值的计算就是确定未来终值的现在价值。

单利现值的计算公式为:

$$P = F/(1 + i \times t) \tag{4-16}$$

【例 4-9】 假设银行存款利率为 10%,为 3 年后获得 20 000 现金,现在应存入银行多少钱?

解:按照单利现值公式有:

$$P = 20\,000/(1 + 10\% \times 3) = 15\,385(\text{元})$$

(2) 复利终值与现值

复利终值是指一定数量的本金在一定的利率下按照复利的方法计算出的若干时期以后的本金和利息。项目成本计算一般采用复利计算。

在复利计算中,设定以下符号:

P:现值;i:利率;F:终值;n:时间(或称期数)。

① 复利终值计算

假定年利率为 i，如果现在存入 P 元，则 n 年后可以得到的钱数 F 为：

$$F = P(1+i)^n \tag{4-17}$$

其中：$(1+i)^n$ 被称为复利终值系数，用符号$(F/P,i,n)$表示。例如，$(F/P,8\%,5)$表示利率为 8%、5 年(期)的复利终值系数。

【例 4-10】 某公司投资一项目，第一年年初一次性投入 180 000 元，预计该项目第一年、第二年、第三年年底收益分别为 50 000 元、80 000 元、120 000 元。假设年利率为 7%，请按复利方法计算该投资额在第三年年底的终值以及各年收益额现值。

解：(1) 计算第三年年底的终值：

$$F = P \times (F/P,i,n) = 180\,000 \times (F/P,7\%,3)$$
$$= 180\,000 \times 1.225 = 220\,500(\text{元})$$

(2) 由公式计算各年收益额现值：

$$P = F \times (P/F,i,n)$$
$$= 50\,000 \times (P/F,7\%,1) + 80\,000 \times (P/F,7\%,2) + 120\,000 \times (P/F,7\%,3)$$
$$= 50\,000 \times 0.9346 + 80\,000 \times 0.8734 + 120\,000 \times 0.8163$$
$$= 46\,730 + 69\,872 + 97\,956 = 214\,558(\text{元})$$

答：按复利方法计算该投资额在第三年年底的终值为 220 500 元，各年收益额现值总额为 214 558 元。

【例 4-11】 某组织投资 100 000 元用于某软件项目开发，在年回报率为 6%的情况下，需要经过多少年才能使现有货币增加 1 倍？

解：依题意，根据复利终值计算公式 $F=P(1+i)^n$ 可得：

$$200\,000 = 100\,000 \times (1+0.06)^n$$

$2=1.06^n$，解该方程，方程两边取对数可得：

$$\log 2 = n\log 1.06$$

$$n = \frac{\log 2}{\log 1.06} \approx 12(\text{年})$$

答：需要经过 12 年才能使现有货币增加 1 倍。

② 复利现值计算

复利现值刚好与复利终值计算相反，即把未来价值折合成今天的价值。其计算公式为：

$$P = F(1+i)^{-n} \tag{4-18}$$

该公式表明了如果 n 年后能收入 F 元钱，那么这些钱的现在的价值是 P。式中：$(1+i)^{-n}$称为复利现值系数，用$(P/F,i,n)$表示。例如，$(P/F,5\%,4)$表示利率为 5%，4 期的复利现值系数。

【例 4-12】 某软件公司计划 4 年后开发一种新的软件产品，需要资金 120 万元，当银行利率为 5%时，公司现在应存入银行的资金是多少？

解：$P = F\times(1+i)^{-n} = 1200\,000\times(1+5\%)^{-4} = 1200\,000\times0.8227$

$=987\,240(\text{元})$

(3) 年金终值与现值

年金是指每隔相等时间间隔收到或支付相同金额的收付款项，如分期偿还贷款、发放养

老金、支付租金、提取折旧等都属于年金收付形式。年金有多种形式，这里只介绍普通年金。

在年金的计算中，设定以下符号：

A：每年收付的金额；i：利率；F：年金终值；P：年金现值；n：时间(期数)

① 年金终值

普通年金终值是指一定时期内每期期末等额收付款项的复利终值之和。

年金终值计算由复利终值公式求得，其公式为：

$$F = A \cdot (1+i)^0 + A \cdot (1+i)^1 + A \cdot (1+i)^2 + \cdots + A \cdot (1+i)^{n-1} \tag{4-19}$$

年金终值也可以由复利终值公式推导而得(从略)，其公式为：

$$F = A \times \frac{(1+i)^n - 1}{i} \tag{4-20}$$

其中：$\frac{(1+i)^n-1}{i}$ 通常称为“年金终值系数”，用符号$(F/A,i,n)$表示。

【例 4-13】 每年在银行存入 4000 元，银行存款利率 5%，到第 4 年年末能得到的资金总额是多少？

解：(1) 用复利终值公式计算年金终值的公式为：

$$\begin{aligned} F &= A \cdot (1+i)^0 + A \cdot (1+i)^1 + A \cdot (1+i)^2 + \cdots + A \cdot (1+i)^{n-1} \\ &= 4000 \times 1 + 4000 \times (1.05)^1 + 4000 \times (1.05)^2 + 4000 \times (1.05)^3 \\ &= 4000 \times (1 + 1.05 + 1.1025 + 1.1576) = 4000 \times 4.31 \\ &= 17\,240(\text{元}) \end{aligned}$$

(2) 用年金终值公式计算：

$$\begin{aligned} F &= A \times \frac{(1+i)^n - 1}{i} \\ &= 4000 \times \frac{(1+5\%)^4 - 1}{5\%} \\ &= 4000 \times 4.31 \\ &= 17\,240(\text{元}) \end{aligned}$$

从以上计算可以看出，通过复利终值计算年金现值比较复杂。

② 年金的现值

普通年金现值是指一定时期内每期期末收付款项的复利现值之和。

根据复利现值的方法计算年金现值 P 的计算公式为：

$$P = A \cdot \frac{1}{(1+i)} + A \cdot \frac{1}{(1+i)^2} + \cdots + A \cdot \frac{1}{(1+i)^{n-1}} + A \cdot \frac{1}{(1+i)^n} \tag{4-21}$$

年金现值公式也可以由复利现值公式推导而得(从略)，其公式为：

$$P = A \cdot \frac{1 - (1+i)^{-n}}{i} \tag{4-22}$$

其中：$\frac{1-(1+i)^{-n}}{i}$通常称为“年金现值系数”，用符号$(P/A,i,n)$表示。

【例 4-14】 某组织分期购买一设备，每年年末支付 10 000 元，分 5 次付清，假设年利率为 5%，则该项分期付款相当于现在一次性支付多少元？

解：本题相当于求每年年末付款 10 000 元，共计支付 5 年的年金现值，可用以下两种计算方式求得年金的现值。

(1) 用复利现值的方法计算年金现值:

$$P = A \cdot \frac{1}{(1+i)} + A \cdot \frac{1}{(1+i)^2} + \cdots + A \cdot \frac{1}{(1+i)^{n-1}} + A \cdot \frac{1}{(1+i)^n}$$
$$= 10\,000 \times [(1.05)^{-1} + (1.05)^{-2} + (1.05)^{-3} + (1.05)^{-4} + (1.05)^{-5}]$$
$$= 10\,000 \times (0.952 + 0.907 + 0.864 + 0.823 + 0.784) = 10\,000 \times 4.33$$
$$= 43\,300(\text{元})$$

(2) 用年金现值公式计算:

$$P = 10\,000 \times (P/A, 5\%, 5) = 43\,300(\text{元})$$

4.6.4 软件工程经济学中的成本效益评价技术

一般来说,人们投资于一项事业的目的是为了在将来获得更大的好处。开发一个新项目也是一种投资,期望将来获得更大的经济效益。经济效益通常表现为减少运行费用或增加收入。但是,投资开发新系统往往要冒一定的风险,系统的开发成本有可能比预计的高,运行效益有可能比预计的差。那么,在什么情况下投资开发新系统更划算?成本/效益分析的目的正是从经济的角度分析开发一个特定的新系统是否划算,是否值得投资。

在经济学领域,发展了许多技术(成本-效益分析、现值分析和风险分析等)来处理有关的决策问题。从风险管理与风险防范的角度来看,只有在效益超过成本的情况下,才考虑让项目继续进行。

1. 净利润

项目的净利润(Net Profit)是在项目的整个生命周期中总成本和总收入之差。

净利润是一项非常重要的经济指标。对于项目的开发者或投资者来说,净利润是获得投资回报大小的基本因素,对于项目管理者而言,净利润是进行经营管理决策的基础。同时,净利润也是评价企业赢利能力、管理绩效以至偿债能力的一个基本工具,是一个反映和分析企业多方面情况的综合指标。

2. 回收期

回收期(Payback Period)表示资金预算项目收回所有成本所需要的时间。用公式可表示为:

$$\text{回收期} = \frac{\text{原始投资额}}{\text{每年现金净流入量}} \tag{4-23}$$

例如,某一软件项目总成本为 20 万元,该项目预期年度收益为 4 万元,则:

回收期=200 000/40 000=5 年

回收期或许是评估投资项目的一个最简便的方法,并且容易为决策人所正确理解。这个方法关注于回收投资成本所需的时间,回收年限越短,方案越有利。如果软件组织希望最小化项目所花的时间,那么通常会选择具备最短偿还期的项目。

事实上,有战略意义的长期投资往往早期收益较低,而中后期收益较高。回收期法优先考虑急功近利的项目,可能导致放弃长期成功的方案。

3. 投资回报率

投资回报率(Return On Investment,ROI)是指通过投资而应返回的价值,企业从一项

投资性商业活动的投资中得到的经济回报。在衡量工程的经济效益时，它是重要的参考数据。其计算公式为：

$$ROI = (平均年利润 / 投资总额) \times 100\% \tag{4-24}$$

ROI 的优点是计算简单；缺点是没有考虑资金时间价值因素，不能正确反映建设期长短及投资方式不同和回收额的有无等条件对项目的影响。只有投资利润率指标大于或等于无风险投资利润率的投资项目才具有财务可行性。

4. 净现值

净现值(Net Present Value，NPV)法是一种项目评价技术。净现值就是净的现在价值。一个投资项目的净现值等于一个项目整个生命周期内预期未来每年净现金流的现值减去项目初始投资支出。然后根据净现值的大小来评价投资方案。净现值为正值，投资方案是可以接受的；净现值是负值，投资方案就是不可接受的。净现值越大，投资方案越好。净现值法是一种比较科学也比较简便的投资方案评价方法。净现值的计算公式为：

$$NPV = 未来收入的总现值 - 所有支出的现值 \tag{4-25}$$

净现值实际上就是计算现金净流量，是一种经过贴现后现金流入量与现金流出量的差额。之所以贴现，是要扣除按设定贴现率所期望的基本投资回报。如果净现值大于零，说明该项目在扣减了基本成本后尚有余额。因此，净现值的经济意义是：投资方案超过投资成本后的超额收益。

【例 4-15】 某投资者准备投资于一个投资额为 20 000 元的软件项目，项目期限 4 年，所期望的投资报酬率为 10%，每年能获取现金流量 7000 元。试计算第 4 年的净现值是多少？项目是否可行？

解：(1) 用复利现值公式计算：

$$\begin{aligned} NPV &= 7000 \times [(P/F,10\%,1) + (P/F,10\%,2) + (P/F,10\%,3) \\ &\quad + (P/F,10\%,4)] - 20\,000 \\ &= (6363 + 5785 + 5259 + 4781) - 20\,000 \\ &= 2188(元) \end{aligned}$$

(2) 用年金现值公式计算：

$$NPV = 7000 \times (P/A,10\%,4) - 20\,000 = 2189.30(元)$$

答：第 4 年的净现值是 2188 元，由于净现值大于零，该项目可行。

净现值指标的决策标准是：如果投资项目的净现值大于零(为正数)，表明当时偿还本息后该项目仍有剩余的收益，接受该项目；如果投资项目的净现值小于零(为负数)，表示该项目收益不足以偿还本息，放弃该项目；当净现值为零时偿还本息后一无所获。如果有多个互斥的投资项目相互竞争，应选取净现值最大的投资项目。

净现值法简便易行，其最大的优点如下。

(1) 适用性强，能基本满足期限相等的互斥投资方案的决策。互斥投资方案是指两个以上投资项目不能同时并存，相互排斥。如生产同一产品可用机器 A 或 B，但采用 A 就不能采用 B，不能同时采用。

(2) 净现值法假定投资项目各期所产生的现金流量，都是以所采用资本成本率作为平均报酬率取得的，比较客观。

(3) 能灵活考虑投资风险。净现值法只要在所采用的贴现率中包括要求的投资风险报酬率,就能有效地考虑投资风险。

本章小结

软件项目的估算是软件项目管理中最困难最重要的活动之一。对项目进行规模、工作量、时间、成本、关键资源进行估算,同时又是制定项目计划和再计划以及跟踪与监督项目必不可少的活动。软件项目中的重点主要在于工作量的估算。如果不能估算出执行一个项目到底需要付出多少工作量和时间,那么也就不可能进行有效的项目计划和管理。

如何根据以往的项目开发经验,预算当前项目的规模、成本、进度(时间)就是估算要解决的问题。软件估算就是如何利用数学模型来根据以往经验做出项目预算的过程,其根本目的是通过对软件项目管理和开发工作量的估算,确认项目开发的成本、开发周期,以作为项目投标、立项和项目开发的依据,同时,通过量化数据的分析与总结,提高软件项目的管理能力和生产率,降低成本和产品研发周期,尽可能地减少因错误估算给企业带来的损失。

软件工程经济学是软件工程学的主要分支之一,它在软件工程项目中起着重要的作用,也是软件工程项目管理人员应该了解与掌握的内容。本章简要介绍了软件工程经济学中一些最基本的内容,可以帮助项目管理者对项目状态进行成本效益分析与决策。

思考与练习

1. 什么是估算?估算具有哪些属性或特征?
2. 相对于传统工程项目,软件项目估算具有哪些特点?其复杂性表现在哪些方面?
3. 软件项目估算涉及哪些内容?
4. 项目规模估算的常用方法有哪些?试比较各方法的优劣。
5. 基于数学模型的工作量估算方法有哪些?
6. 什么是 COCOMO 模型?它如何用不同层次的模型来反映软件项目不同程度的复杂性?
7. 基本 COCOMO 模型的 3 种应用开发模式各适用什么样的开发环境?
8. COCOMO Ⅱ模型与基本 COCOMO 相比较有哪些改进?COCOMO Ⅱ的 3 个生命周期模型是指什么?
9. 方差与标准差在三点估算技术中如何计算,对于估算项目成本与规模有何意义?
10. 试根据 Putnam 模型给出的公式,分析软件项目的工作量、软件开发时间和程序代码长度 3 者之间的关系。
11. 软件项目成本由哪些方面组成?常用的估算方法有哪些?
12. 请解释静态单变量模型、静态多变量模型与动态多变量模型有何区别。
13. 一些非常大型的软件项目都有数以百万计的程序代码行;试说明成本估算模型对于这种系统的重要意义。为什么对这些成本估算模型的一些假设对非常大型的软件系统可能无效?

14. 无论使用什么估计方法，成本估算有其固有的风险，试总结出能降低成本估算风险的几种方法。

15. 项目进度获取值分析有哪些基本度量？其含义是什么？

16. 什么是软件工程经济学？它研究的基本问题是什么？

17. 什么是资金的时间价值？请解释现值、终值、单利、复利的基本概念。

18. 什么是净现值？如何根据净现值对项目进行评价？

19. 已知有一个典型的软件项目的记录，开发人员 $M=3$ 人，其代码行数=12.1KLOC，工作量 $E=24$PM，成本 $S=168\,000$ 美元。试计算开发该软件项目的生产率 P 和每代码行平均成本 C。

20. 使用 COCOMO Ⅱ模型来估算一个软件系统所需的工作量，该系统产生 12 个屏幕、10 个报表，需要大约 80 个软件构件。假定该软件具有“中等”复杂度和“正常开发者环境”成熟度，复用的百分比为 40%。试完成以下计算：

(1) 计算项目的 NOP。

(2) 进行工作量估算(人月)。

(3) 假设软件人员劳动力平均价格是每月 5000 元人民币，求每个 NOP 的成本。

21. 根据下面的信息域特性值：

外部输入数	外部输出数	外部查询数	内部接口数	外部文件数
10	12	15	8	8

假设各项加权因子均为 5，不考虑复杂度调整值 $\left(即：\sum F_i = 0\right)$，试完成以下计算：

(1) 计算项目的功能点的总计数值 FP。

(2) 设平均生产率为 10FP/pm，软件人员劳动力平均价格是每月 5000 元人民币，求每个 FP 的成本。

(3) 根据 FP 值计算总的项目成本，并进行工作量估算(人月)。

22. 在人员和时间之间的关系是高度非线性的。使用 Putnam 的软件公式，编制一个表，反映软件项目中人员数量与项目持续时间之间的关系，该项目需要 50 000 LOC 和 15 人年的工作量(生产率参数为 5000，且 $B=0.37$)。假定该软件必须在 24 个月和加减 12 个月的时间期限内交付。

23. 表 4-13 是某公司的一个项目的历史数据，假设评价时间为 2008 年 4 月 1 日，试利用获得值分析公式计算如下指标：

表 4-13　项目历史数据

工作任务	估计工作量成本人天	实际工作量成本	估计完成日期	实际完成日期
1	5	10	2008-1-25	2008-2-1
2	20	15	2008-2-15	2008-2-15
3	50	6	2008-5-15	
4	40	50	2008-4-15	2008-4-1
5	60	50	2008-7-1	
6	80	70	2008-9-1	

(1) 预计工作的预算成本 BCWS。

(2) 已完成工作的预算成本 BCWP。

(3) 完成工作的预算成本 BAC。

(4) 已完成工作的实际成本 ACWP。

24. 假设以 10%的年利率借得 30 000 元,投资于某个生命周期为 10 年的项目,为使该投资项目成为有利的项目,每年至少应收回的现金数额为多少?

25. 某个固定资产投资项目需要原始投资 1000 万元,有 A、B、C、D 4 个互相排斥的备选方案可供选择,各方案的净现值指标分别为 420.89 万元,511.72 万元,620.60 万元和 556.26 万元。试按净现值法进行比较决策。

26. 某人拟存入一笔资金以备 3 年后使用。假定银行 3 年期存款年利率为 5%,3 年后需用的资金总额为 34 500 元,则在单利计算情况下,目前需存入的资金是多少元?

27. 某公司准备购置一台设备,有甲、乙两种可供选择,甲设备比乙设备高 5030 元,但每年使用费可节约 600 元。该设备可以使用 10 年,假设年利率为 6%。该公司应选择使用哪一种设备?

第5章 软件质量管理

质量是一个难以捉摸的术语，软件的质量更难以捉摸。

——Robert L. Glass

软件的嵌入式应用使软件无处不在，涉及日常生活中的方方面面。软件在手机、电视等家用电器中，在所乘坐的交通工具中，在银行及股市系统中，在ATM机中，在医院用于监控和抢救生命的设备中，发挥着越来越重要的作用。在这些系统中的任何一个缺陷都会对我们的生活甚至整个社会产生影响。

许多软件工程教科书为了强调质量的重要性，总是要列举一些历史上发生过的重大软件质量事故，如航天飞机爆炸，核电站失事，爱国者导弹发生故障等。这些事故的确不是危言耸听，它们给人们敲响了质量的警钟。当然，大多数普通软件的缺陷并不会造成如上这样的重大损失，否则软件程序员就成了风险极大的职业了。

软件产品质量的评价、管理和控制都是非常困难的。一方面，由于人们对软件生产的认识还不够充分；另一方面，软件开发过程相当复杂，缺乏直观性。在产品质量控制中，质量管理活动是关键，这就使软件产品质量保证与控制活动非常必要。

5.1 软件质量及其特性

5.1.1 难以定义和度量的软件质量

软件质量是一个模糊的、令人捉摸不定的概念。我们常常听说：某软件功能全面，易学易用；某软件界面美观，操作方便等。这些模糊的语言实在不能算作是软件质量评价，特别不能算作是软件质量科学的定量的评价。

一般而言，质量是一组固有特性满足要求的程度，是产品或服务满足规定或潜在需要的特征和特性的总和。它既包括有形产品，也包括无形产品；既包括产品内在的特性，也包括产品外在的特性，此外，也包括了产品的适用性和符合性的全部内涵。

软件质量是一个复杂的概念，不同的人从不同的角度来看软件质量问题，会有不同的理解。从用户的角度看，质量就是满足客户的需求；从开发者的角度看，质量就是与需求说明保持一致；从产品的角度看，质量就是产品的内在特点；从价值的角度看，质量就是客户是否愿意购买。

有多种关于软件质量的定义。例如，按照ANSI/IEEE Std1061-1992中的标准，软件质

量定义为“与软件产品满足规定的和隐含的需求的能力有关的特征或特性的全体”[①]。

根据 ISO 8402 术语规定，质量就是：“反映实体满足明确和隐含需要的能力的特性总和。”软件质量是指：“对用户在功能和性能方面需求的满足、对规定的标准和规范的遵循以及规范软件某些公认的应该具有的本质。”

国际标准化组织 ISO 在质量特性国际标准 IEC 9126 中将软件质量定义为：反映软件产品满足规定需求和潜在需求能力的特征和特性的总和[②]。

从以上定义可知，软件质量反映了以下 3 个方面的问题。

(1) 软件需求是度量软件质量的基础，不符合需求的软件就不具备质量。

(2) 规范化的标准定义了一些开发准则以指导软件开发，如果不遵守这些开发准则，软件质量就得不到保证。

(3) 往往会有一些隐含的需求没有明确地提出来。例如，软件应具备良好的可维护性。如果软件只满足那些精确定义了的需求而没有满足这些隐含的需求，软件质量也不能保证。

5.1.2 软件质量特性

质量特性是指实体所特有的性质，它反映实体满足需求的能力。软件质量特性反映了软件的本质。讨论一个软件的质量，问题最终要归结到定义软件的质量特性上。而定义一个软件的质量，就等价于为该软件定义一系列质量特性。

通常，可以用软件质量模型来描述影响软件质量的特性。质量模型既考虑了软件产品的需求也考虑了过程的需求，确定了适应于软件产品需求的质量特性、特征和度量，有助于软件质量不同概念的标准化。

人们已提出多种有关软件质量的模型，它们共同的特点是把软件质量特性定义成分层模型。在这种分层的模型中，最基本的叫做基本质量特性，它可以由一些子质量特性定义和度量。二次特性在必要时又可由它的一些子质量特性定义和度量。下面是几个影响较大的软件质量模型。

1. Boehm 质量模型

1976 年，Boehm 等人提出了定量地评价软件质量的概念，软件产品质量主要应从以下 3 个方面来评价。

(1) 软件的可使用性。

(2) 软件的可维护性。

(3) 软件的可移植性。

为此，Boehm 给出了 60 个质量量度公式说明怎样用来评价软件质量。从模型可知，他们把软件质量概念分解为若干层次，对于最低层的软件质量引入数量化的概念，以此求得对软件质量的整体评价。

---

① Software Product Evaluation-Quality Charateeristies and Guideline for Their Use. ISO/IEC Standard ISO-9126, 1991

② Software Quality Charateeristies. ISO/TC97/SC7/WG3/1985-1-30/N382

2. McCall 质量模型

McCall 模型也是最早提出的软件质量模型之一。这是一个包括软件质量要素、评价准则和度量 3 个层次的软件质量度量模型框架。其上层是外部观察的特性，中间层是评价准则，下层是软件内在的特性。McCall 又给出了 11 个软件质量特性，如表 5-1 所示。这些特性分别面向软件产品的运行、修正和转移，它们表现了系统可见的行为化特征，如图 5-1 所示。

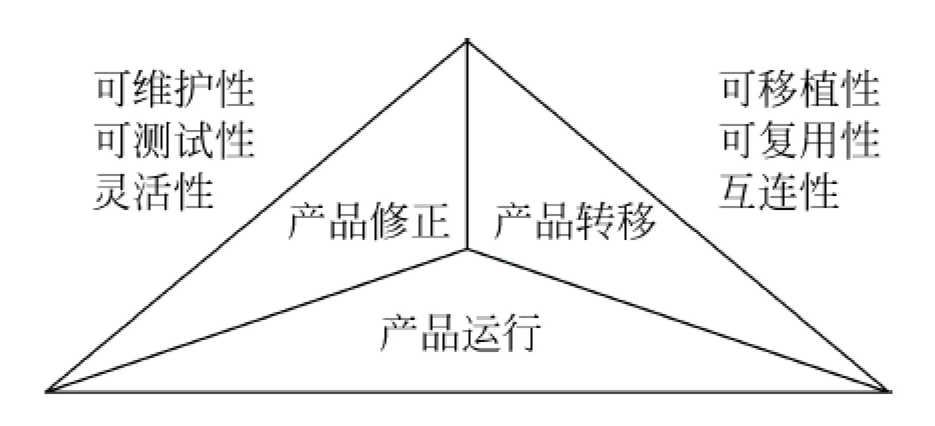


图 5-1　McCall 的 11 个软件质量要素

表 5-1　软件质量的 11 个特性描述

属　性	描　　述
正确性	在预定环境下，软件满足设计规格说明及用户预期目标的程度。它要求软件没有错误
可靠性	软件按照设计要求，在规定时间和条件下不出故障，持续运行的程度
效率	为了完成预定功能，软件系统所需的计算机资源的多少
完整性	为了某一目的而保护数据，避免它受到偶然的，或有意的破坏、改动或遗失的能力
可使用性	对于一个软件系统，用户学习、使用软件及为程序准备输入和解释输出所需工作量的大小
可维护性	为满足用户新的要求，或当环境发生了变化，或运行中发现了新的错误时，对一个已投入运行的软件进行相应诊断和修改所需工作量的大小
可测试性	测试软件以确保其能够执行预定功能所需工作量的大小
灵活性	修改或改进一个已投入运行的软件所需工作量的大小
可移植性	将一个软件系统从一个计算机系统或环境移植到另一个计算机系统或环境中运行时所需工作量的大小
复用性	一个软件(或软件的部件)能再次用于其他应用(该应用的功能与此软件或软件部件所完成的功能有联系)的程度
互连性	连接一个软件和其他系统所需工作量的大小。如果这个软件要与其他系统通信，或要把其他系统纳入到自己的控制之下，必须有系统间的接口，使之可以联结

McCall 等认为，特性是软件质量的反映，软件属性可用做评价准则，定量化地度量软件属性可了解软件质量的优劣。McCall 定义的软件质量要素评价准则共 21 种，包括可审查性、准确性、通信通用性、简明性等。

软件质量要素 F_j 的值可用式(5-1)计算：

$$F_j = \sum_{k=1}^{21} C_k M_k \quad 1 \leqslant j \leqslant 11, \quad 1 \leqslant k \leqslant 21, \quad \sum_{k=1}^{21} C_k = 1 \tag{5-1}$$

其中：M_k 是软件质量要素 F_j 对第 k 种评价准则的测量值；C_k 是相应的加权系数。

但是，对以上各个质量特性直接进行度量是很困难的，在有些情况下甚至是不可能的。因此，McCall 定义了一些评价准则，使用它们对反映质量特性的软件属性分级，以此来估计软件质量特性的值。McCall 将评分准则划分为 0～10 级。0 级最低，10 级最高。因此，M_k 的取值可以是 0，0.1，0.2，0.3，…，1.0。

3. ISO/IEC 9126 质量模型

ISO/IEC 9126 质量模型是另一个著名的质量模型，最新版本为 ISO/IEC 9126:2001，它描述了包括软件内部质量和外部质量在内的 6 个特征。它们还可以再继续分成 21 个子特征，这些子特征在软件作为计算机系统的一部分时会明显地表现出来，并且会成为内在的软件属性的结果。ISO/IEC 9126 质量模型如图 5-2 所示。

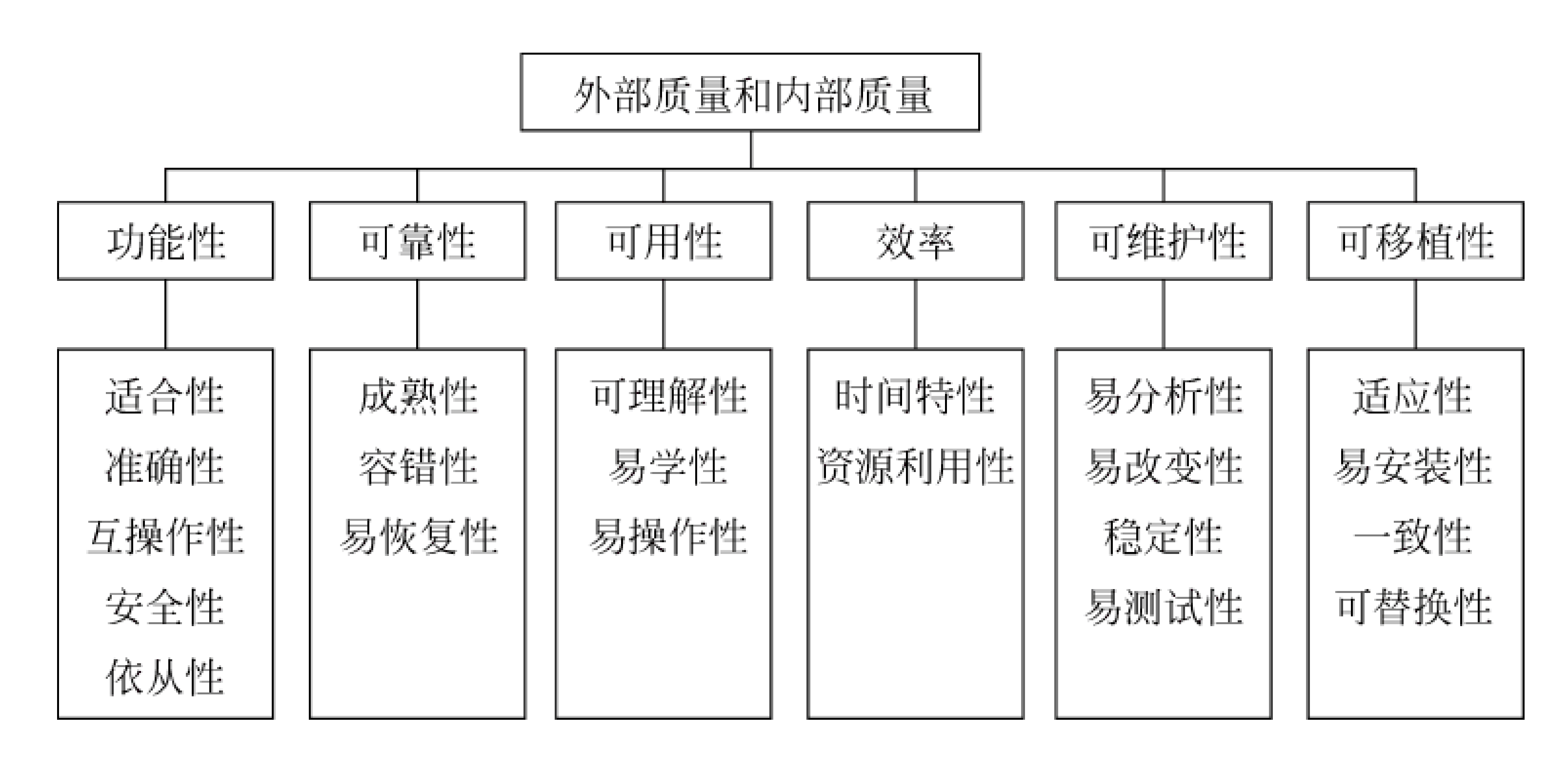


图 5-2 ISO/IEC 9126 质量模型

ISO/IEC 9126 质量模型的 6 个质量特性描述如下。

(1) 功用性(Functionality)，即软件是否满足了客户功能要求。

(2) 可靠性(Reliability)，即软件是否能够一直在一个稳定的状态上满足可用性。

(3) 可用性(Usability)，即衡量用户能够使用软件需要多大的努力。

(4) 效率(Efficiency)，即衡量软件正常运行需要耗费多少物理资源。

(5) 可维护性(Maintainability)，即衡量对已经完成的软件进行调整需要多大的努力。

(6) 可移植性(Portability)，即衡量软件是否能够方便地部署到不同的运行环境中。

5.1.3 软件质量保证及其活动

美国项目管理协会(PMI)对项目质量保证的定义是：“项目质量保证是一种有目的、有计划和有系统的活动。”IEEE 标准 729 中有关质量保证的定义归纳为：“质量保证是为了确保项目或产品符合基本技术需求，而必须采取的有计划的、系统的全部动作的模式。”CMM 对软件质量保证是这样描述的：“软件质量保证的目的是为管理者提供有关软件过程和产品的适当的可视性。它包括评审和审核软件产品及其活动，以验征其是否遵守既定的规程和标准，并向有关负责人汇报评审和审核的结果。”

软件质量保证(Software Quality Assurance，SQA)是一个复杂的系统，它采用一定的技术、方法和工具，来处理和调整软件产品满足需求时的相互关系，以确保软件产品满足或超过在该产品的开发过程中所规定的标准。

SQA 是一种应用于整个软件过程的活动，其工作内容主要包括以下 6 类。

(1) 与 SQA 计划直接相关的工作。SQA 在项目早期要根据项目计划制定与其对应的 SQA 计划，定义出各阶段的检查重点，标识出检查、审计的工作产品对象，以及在每个阶段

SQA 的输出产品。

(2) 参与项目的阶段性评审和审计。在 SQA 计划中通常已经根据项目计划定义了与项目阶段相应的阶段检查,包括参加项目在本阶段的评审和对其阶段产品的审计。SQA 参与评审是从保证评审过程有效性方面入手,如参与评审的人是否具备一定资格、是否规定的人员都参加了评审、评审中是否对被评审的对象的每个部分都进行了评审并给出了明确的结论等。

(3) 对项目日常活动与规程的符合性进行检查。这部分的工作内容是 SQA 的日常工作内容。由于 SQA 独立于项目组,如果只是参与阶段性的检查和审计很难及时反映项目组的工作过程,所以 SQA 也要在两个阶段点之间设置若干小的跟踪点,来监督项目的进行情况,以便能及时反映出项目组中存在的问题,并对其进行追踪。

(4) 对配置管理工作的检查和审计。SQA 要对项目过程中的配置管理工作是否按照项目最初制定的配置管理计划进行监督,包括配置管理人员是否定期进行该方面的工作、是否所有人得到的都是开发过程产品的有效版本。

(5) 跟踪问题的解决情况。对于评审中发现的问题和项目日常工作中发现的问题,SQA 要进行跟踪,直至解决。

(6) 度量和报告机制。

以上活动需由 SQA 小组产生文档,并形成质量报告,发送给软件项目经理、项目开发人员和所有相关人员。

5.2 软件配置管理

软件配置管理的根源可以追溯到 20 世纪 60 年代或者 70 年代的制造业配置控制问题,特别是在生产复杂硬件的领域,例如汽车和飞机制造业中,产品部件的材料清单、产品部件标识、装配规程和测试(检验)规程等,这些规程可以用于指导产品配置项的设计、集成和测试的过程。在这一时期,随着许多大型管理信息系统的开发,开始出现了对软件开发过程实施配置管理的需求。

软件配置管理(Software Configuration Management,SCM)是控制软件系统演进的学科,软件配置管理是指通过执行版本控制、变更控制等规程,以及使用合适的配置管理软件,来保证所有配置项的完整性和可跟踪性。配置管理是对工作成果的一种有效保护。

软件配置管理是一项质量管理活动,它贯穿于整个软件生命周期,为软件研发提供了一套管理办法和活动原则。软件配置管理无论是对于软件企业管理人员还是研发人员都有着重要的意义。

软件配置管理的主要任务可归结为以下几条。

(1) 制定项目的配置计划。

(2) 对配置项进行标识。

(3) 对配置项进行版本控制。

(4) 对配置项进行变更控制。

(5) 正式技术复审。

(6) 向相关人员报告配置的状态。

5.2.1 制定项目的配置计划

在项目立项后，项目组就需要开始制定项目管理计划，编写规范的配置管理计划就是要明确如何实施配置管理活动。这时配置管理组开始参与和考虑配置管理计划的制定。计划应该确保短小、清晰并且具有可操作性，便于人们阅读和理解，在计划文档中只写进和配置管理有关联的内容。

该计划的内容包括要执行的配置管理活动，即对配置项的标识、控制、状态记录、审核，对所需的组织及其各自的职责编制配置管理里程碑。

配置计划涉及以下内容。

1. 组织与职责

明确指派负有下列职责的各类人员：负责配置管理计划的审批、实施与更改跟踪的软件配置管理经理，在整个软件生命过程中按照配置管理计划执行配置管理活动的软件配置管理负责人。

2. 配置标识

标识需要识别每一基准配置项，列出要标识的所有配置项及其相应的标识规范。标识涉及下列信息：何时及如何提交、批准人和验证人、目的、提交方式（软件或文档）及版本号。还要对软件工具、硬件设备等进行标识。

3. 配置库内容

标识和控制规范、文档库的数目及类型、备份及作废计划和程序、任何损失的恢复过程、文档保留程序、什么文档要保留和谁保留及保留多长时间、信息是在线还是脱机保留以及保留介质。

4. 变更控制

明确指派负责更改控制的组织。当需要变更时，如何控制这些变更，有什么样的评审机制、审批机制，来确保基准配置项的更改依据更改控制程序进行。

5. 配置状态报告

项目日常运作开始后，要定期生成报告，报告的内容必须是项目相关人员想要了解的信息。

6. 配置审核

指派负责配置审核的软件质量保证负责人及配合人员，列举要进行审核的配置项。

在软件项目整个开发期间，必须成立软件配置管理小组负责配置管理工作。软件配置管理小组和软件配置管理人员必须共同担负起整个项目的软件配置管理工作，检查和督促本计划的实施，以确保完全遵守本计划规定的所有要求。

5.2.2 软件配置项及其标识

1. 软件配置项

配置标识或者又称为配置需求,包括标识软件系统的结构,标识独立部件,并使它们是可访问的。配置标识的目的,是在整个生命周期中标识系统各部件并提供对软件过程及其软件产品的跟踪能力。它回答:什么是受控的和怎样受控?

在软件开发过程中生成各种制品的总和叫做这个项目的软件配置[1]。软件配置项(SCI)包括:计算机程序,如源代码和可执行程序;与计算机程序相对应的各种文档,如需求、设计规约(文档)、测试用例等;计算机数据,包括计算机程序中包含的数据和系统初始化数据。所有配置项应由配置系统管理。软件配置项的组成如图5-3所示。

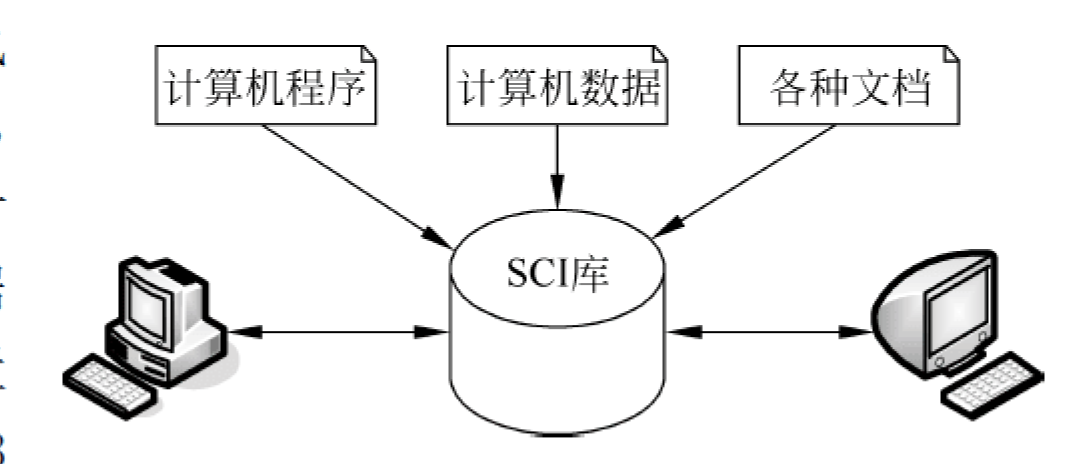


图5-3 软件配置项的组成

2. 软件配置项标识

制定适当的命名规则是配置标识的重要工作。命名不能任意、随机地进行,命名要求具有:

(1) 唯一性:目的在于避免出现重名,造成混乱。

(2) 可追溯性:使命名能够反映命名对象间的关系,每个可用一组信息来唯一地标识。

所有的软件配置项均可以当做对象来标识。例如,基本对象可以是一段需求规格说明、一个模块的源代码清单、一组测试用例等。将这些正文单元命名后作为基本对象,再将基本对象按内部逻辑进行组合,就形成复合对象。复合对象表示的是一本设计规格说明书、一个完整的源程序清单。

3. 配置项的状态

配置项一般有3种状态:草稿(Draft)、正式发布(Released)和正在修改(Changing)。配置项状态变迁如图5-4所示[2]。配置项刚建立时其状态为草稿。配置项通过评审(或审批)后,其状态变为正式发布。此后若更改配置项,必须依照“变更控制规程”执行,其状态变为正在修改。当配置项修改完毕并重新通过评审(或审批)时,其状态又变为正式发布,如此循环。

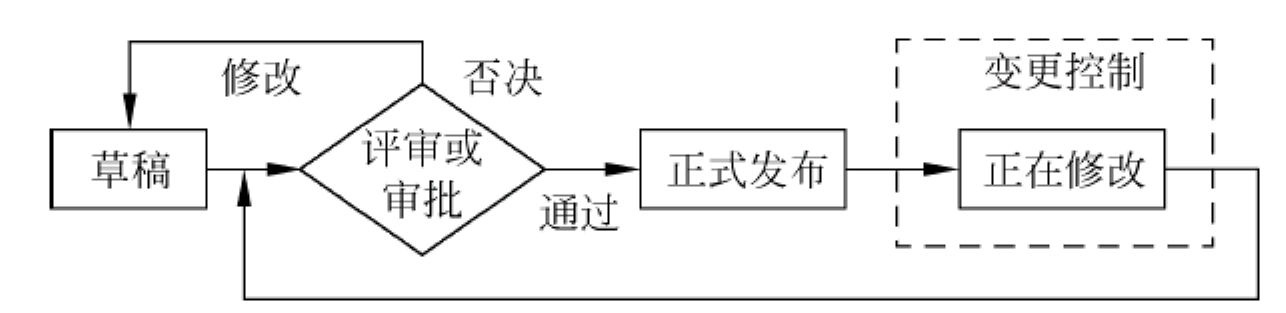


图5-4 配置项状态变迁图

① Roger S. Pressman. 软件工程:实践者之路. 第五版. 北京:清华大学出版社,2001

② 林锐. 软件工程与项目管理解析. 北京:电子工业出版社,2003

4. 基线

配置项的识别是配置管理活动的基础，也是制定配置管理计划的重要内容。为了在不严重阻碍合理变化的情况下来控制变化，软件配置管理中引入了基线(Baseline)这一概念。IEEE 定义基线如下[①]："已经通过正式复审和批准的某规约或产品，它因此可以作为进一步的基础，并且只能通过正式的变化控制过程的改变。"

基线是软件开发的里程碑，其标志是有一个或多个软件配置项的交付，并且这些配置项已经过正式技术复审而获得认可。在软件配置管理中，运用基线概念的一个重要原则是：在软件配置项变成基线前，变化可以迅速而非正式地进行，然而，一旦基线已经建立，我们就得像通过一个单向开的门那样，变化可以进行，但是，必须应用特定的、正式的规程来评估和验证每个变化。这就是说，在软件开发进程中，开发者有权对本阶段的产品进行更改；一旦阶段产品通过复审成为基线 SCI 之后，就应该将它交给配置管理人员去控制，任何人(包括研制该阶段产品的人员)需要对它更改时，都要经过正式的审批手续。正是这种对基线 SCI 的连续控制与跟踪，保证了软件配置的完整性与一致性。

所有基线的 SCI 被放置到项目配置数据库中，这样便于对 SCI 进行检索、提取、修改等配置信息的处理和维护(图 5-5)。

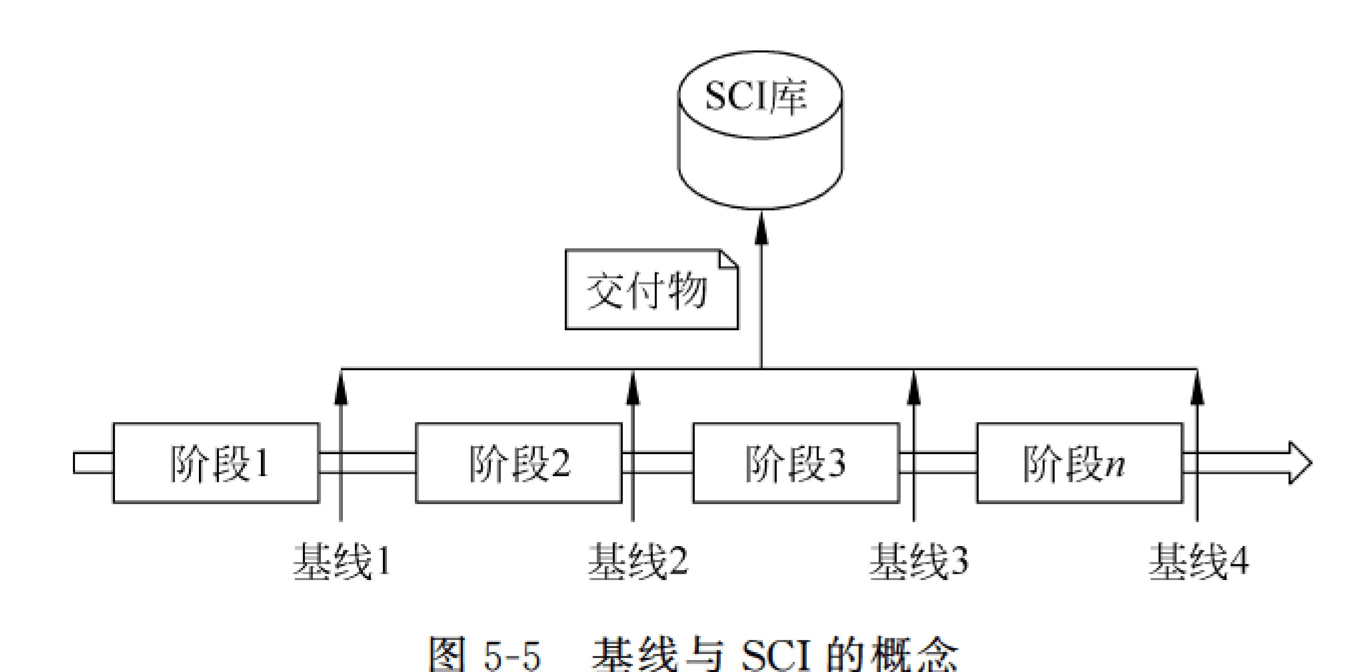


图 5-5 基线与 SCI 的概念

5.2.3 版本控制

版本控制(Revision Control)是全面实行软件配置管理的基础，是对系统不同版本进行标识和跟踪的过程，所有置于配置库中的元素都应自动予以版本的标识，并保证版本命名的唯一性和状态的一致性。其目的是便于对版本加以区分、检索、跟踪或回溯，以区别各个版本之间的关系。

1. 版本表示及演化树

软件系统的版本号由 3 部分构成，即主版本号＋次版本号＋修改号。主版本号 1 位，只有当系统在结构和功能上有重大突破改进后才发生变化；次版本号有 2 位；修改号 8 位，采用提交时的日期，当系统进行任何修改后，包括数据库结构发生变化时，修改号都要随之改变。例如：Ver3.31.20090901。

---

① IEEE Std. 610.12-1990

一个系统的不同版本演化如同一个树结构，最简单的版本只有一个分支，也就是版本的主干，复杂的版本树（如并行开发下的版本树）除了主干外，还可以包含很多的分支，分支可以进一步包含子分支，如图 5-6 所示。树中的每一个结点均是一个 SCI 的集合，即是软件的一个完整版本。根结点是原始版本，叶结点代表最新版本。每个版本有一组源代码、文档、数据等，并且每个版本可以由多种不同的变体组成。

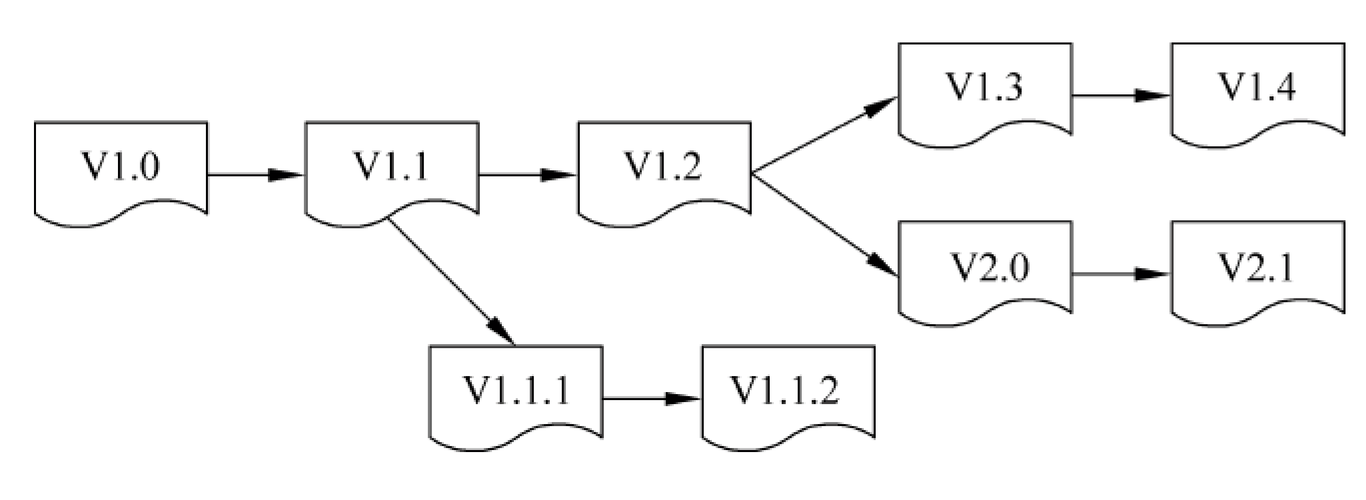


图 5-6　版本变迁的演化树

对于一个采用版本控制进行软件开发的多人开发团队而言，其一般的开发方式是：采用服务器/客户端的形式，在上面分别安装版本控制工具的服务器和客户端版本，软件放在服务器上为大家所共享，开发人员在客户端从服务器上将软件的相关部分下载到本地，进行修改，改动结果最终提交到服务器上。由于采用服务器/客户端方式，尽管开发人员可以在自己的本地留有备份，但最终唯一有效的只有服务器端的那个原始备份。这在一定程度上可以解决一致性问题和冗余问题。

2. 版本控制系统

通过文件名识别版本，对于小型项目或者单个文件也许可行，但是对于软件开发来说，是不适用的。大型的、频繁修改的、多人编写的软件项目，需要一个版本控制系统（简称 VCS），追踪文件的变化，避免出现混乱。

版本控制系统解决了几个问题。首先，它们提供了一种有组织的方式来跟踪对一个文件的修改历史，从而让人理解改动文件的来龙去脉，并且能够恢复早先的版本。其次，它们把版本的概念扩展到了单个文件的层次之外。有关系的一组文件可以按其相互间的依赖性一起指定版本。最后，版本控制系统能够协调多个开发人员之间的活动，使竞争条件（Race Condition）不会因任何人的改动永久丢失，这样一来，多个开发人员做出的不兼容的改动不会同时起作用。

目前有许多商业与开放源码的版本控制系统。常用的最简单的版本控制系统叫做 RCS，名字正好就是版本控制系统（Revision Control System）。另一种常用软件为并发版本系统（Concurrent Versions System，CVS）的开放源代码系统。它支持一种分布式的模型（配合远程服务器来使用），而且更好地支持了多位开发人员的协同工作（图 5-7）。

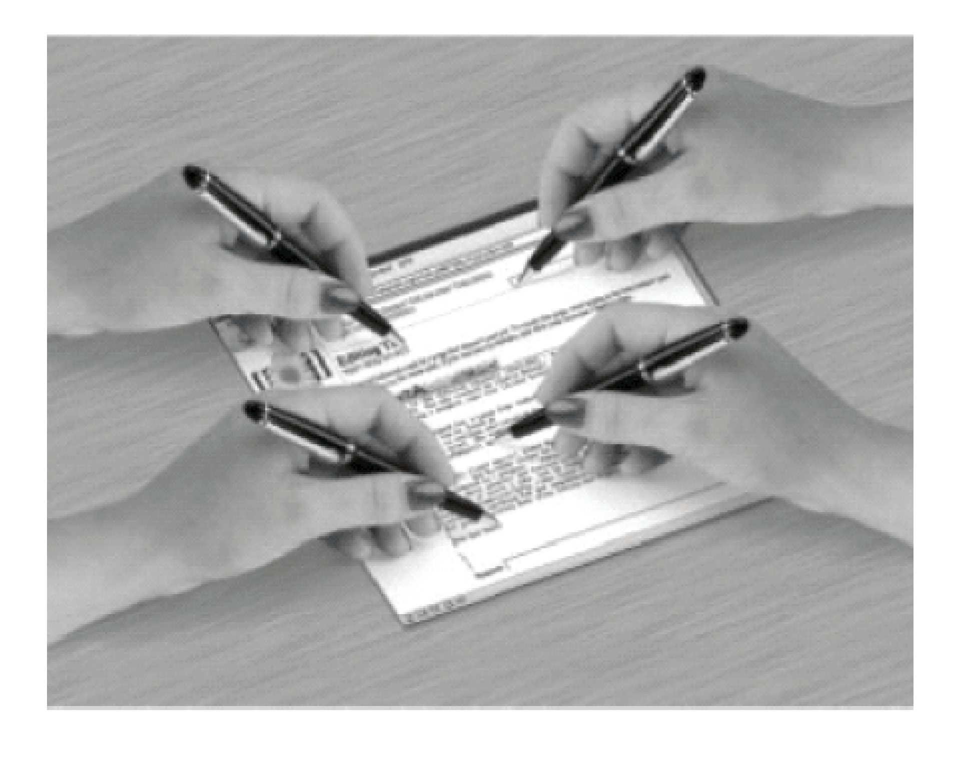


图 5-7　版本控制系统可支持多个开发人员的协同工作

3. 系统版本的发布管理

发布是指经过测试并导入实际应用环境的新增或改进的配置项的集合。发布的类型主要包括全发布、增量发布和包发布 3 种。全发布是指同时构建、测试、分发和实施发布单元的所有组成组件的发布方式。增量发布是指仅仅对自上次全发布或者增量发布以来发布单元中实际发生变化或新增的那些配置项进行发布的方式。包发布是指将一组软件配置项以包的形式一起导入实际运作环境的发布方式。

系统的发布版本是分发给客户的系统版本。系统发布版本管理者负责决定系统发布给客户的时间，管理创建版本的过程和分发渠道，编制发布版本的文档，保证在需要时重新制作与已发布版本完全相同的系统。

系统的发布版本不仅仅是这个系统的可执行代码，发布版本还包括以下内容。

(1) 配置文件。定义对于特定安装，发布版本应该如何配置。

(2) 数据文件。是成功进行系统操作所必需的。

(3) 安装程序。用来帮助在目标硬件上安装系统。

(4) 电子和书面文档。用于系统说明。

(5) 包装和相关的宣传。指为发布版本所做的工作。

4. 发布版本的创建

发布版本的创建是创建包含系统发布版本的所有组件在内的文件和文档集合的过程。程序的可执行代码和所有相关数据文件都必须收集并确认。对不同的硬件和操作系统要写出配置描述，对需要配置自己的系统的客户要为其准备好使用说明书，如果发布了可在机器上阅读的手册，其电子备份必须与软件一起保存起来，还要写出安装程序的脚本。最后，当所有的信息齐备时，准备好发布版本母盘以备发布之用。

目前系统发布版本常用的发布介质是 CD-ROM 光盘。除此之外，许多软件产品也在因特网上发布，允许客户下载这些软件。

5.2.4 变更控制

变更控制(Change Control)的目的并不是控制变更的发生，而是对变更进行管理，确保变更有序进行。对于软件开发项目来说，发生变更的环节比较多，因此变更控制显得格外重要。它回答：受控产品怎样变更？谁控制变更？何时接受，恢复，验证变更？

配置变更控制包括在软件生命周期中控制软件产品的发布和变更，目的是建立确保软件产品质量的机制。配置状态统计包括记录和报告变更过程，目标是不间断记录所有基线项的状态和历史，并进行维护。它解决以下问题：系统已经做了什么变更？此问题将会对多少个文件产生影响？配置变更控制是针对软件产品，状态统计针对软件过程。因此，二者的统一就是对软件开发(产品、过程)的变更控制。

变更控制的目的是防止配置项被随意修改而导致混乱。对于大型的项目，鉴于配置管理的重要性和复杂性，机构应当设立变更控制委员会(Change Control Board，CCB)。CCB 是一个项目主要的管理机构组织，最小应该由下面几部分组成：高层经理、项目经理(技术负责人)、配置管理负责人、质量保证负责人、测试负责人。

对于配置库中的各个基线控制项，应该根据其基线的位置和状态来设置相应的访问权限。一般来说，对于基线版本之前的各个版本都应处于被锁定的状态，如需要对它们进行变更，则应按照变更控制的如下流程来进行操作(图5-8)。

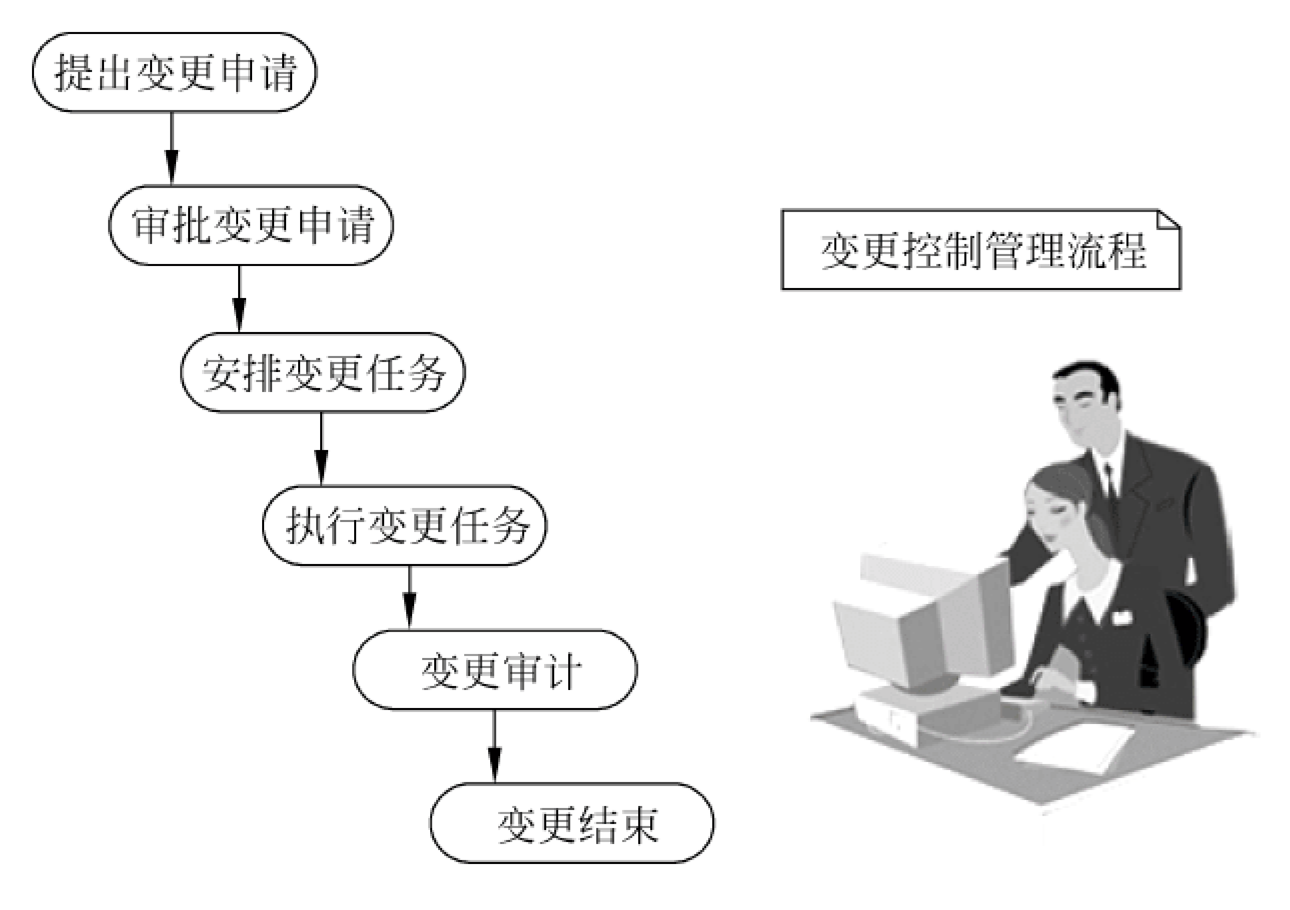


图5-8　变更控制管理流程

(1) 提出变更申请。变更申请人向CCB提交变更申请，重点说明“变更内容”和“变更原因”。

(2) 审批变更申请。CCB负责人(或项目经理)审批该申请，分析此变更对项目造成的影响。如果同意变更，则转向第(3)步，否则终止。

(3) 安排变更任务。CCB指定变更执行人，安排他们的任务。CCB需要和变更执行人就变更内容达成共识。

(4) 执行变更任务。变更执行人根据CCB安排的任务，修改配置项。CCB监督变更任务的执行，如检查变更内容是否正确、是否按时完成工作等。

(5) 变更审计。复审(审计)所有配置项的变化。

(6) 变更结束。当所有变更后的配置项都通过了技术评审或审批，这些配置项的状态从正在修改变为正式发布，并把本次变更通知所有相关人员，本次变更活动结束。

对于普通的小型软件项目而言，CCB这个概念难以落实。出于务实和提高效率，变更控制让项目经理或者配置管理员决定就可以了。

5.2.5　正式技术复审

1. 为什么要进行评审

软件评审(或复审)是软件工程过程中的“过滤器”。复审被用于软件开发过程中的多个不同的点上，起到发现错误(进而引发排错活动)的作用。软件复审起到的作用是“净化”分析、设计和编码中所产生的软件工作产品。IEEE Std 1028-1988定义：评审是对软件元素或者项目状态的一种评估手段，以确定其是否与计划的结果保持一致，并使其得到改进。

技术评审(Technical Review)最初是由IBM公司为了提高软件质量和提高程序员生产率而倡导的。技术评审的目的是尽早地发现工作成果中的缺陷，并帮助开发人员及时消除缺陷，从而有效地提高产品的质量。技术评审方法已经被业界广泛采用并收到了很好的效果，它被普遍认为是软件开发的最佳实践之一。根据IBM的统计数据显示：在大多数企业

的产品开发中,2/3 的缺陷都是在需求和设计阶段引入的。因此,通过评审尽早发现的缺陷的修复成本远低于在产品开发后期测试中发现的缺陷的修复成本。

软件产品被称为"无形"的产品,评审时难度较大。在评审时不能只对最终的软件代码进行评审,还要对软件开发计划、标准、过程、软件需求、软件设计、数据库、手册以及测试信息等进行评审,即对软件过程中产生的所有配置项进行评审。选择评审方法最有效的标准是:对于最可能产生风险的工作成果,要采用最正式的评审方法。

2. 正式技术复审

正式技术复审(Formal Technical Review,FTR)是一种由软件工程师进行的软件质量保证活动。FTR 的目标如下:

(1) 在软件的任何一种表示形式中发现功能、逻辑或实现的错误。

(2) 证实经过复审的软件的确满足需求。

(3) 保证软件的表示符合预定义的标准。

(4) 得到以一种一致的方式开发的软件。

(5) 使项目更易于管理。

由于 FTR 的进行使大量人员对软件系统中原本并不熟悉的部分更为了解,因此,FTR 还起到了提高项目连续性和培训后备人员的作用。

FTR 实际上是一类复审方式,包括走查(Walkthrough)、审查(Inspection)、轮查(Round-robin Review)以及正式的同行评审(Peer Reviews)等技术评估。每次 FTR 都以会议形式进行,只有经过适当的计划、控制和参与,FTR 才能获得成功。

(1) 走查。由生产者启动和主持评审,生产者向评审者展示文档。其优点是启动快,成本低,缺点是容易被生产者误导过程。

(2) 轮查。生产者向评审者进行简要介绍,但不参加评审过程;评审者独立进行评审,并记录发现结果、准备报告。

(3) 同行评审。评审者与生产者是地位平等的同行或专家,是最为结构化的评审方法;可以作为同行之间学习和分享经验的机会。

3. FTR 的过程

FTR 的焦点是某个工作产品——软件的一部分(如一部分需求规约、一个模块的详细设计、一个模块的源代码清单)。开发这一产品的"生产者"通知项目管理者工作产品已经完成,需要进行复审。项目管理者与"复审主席"联系,主席负责评估工作产品是否准备就绪,创建副本,并将其分发给 2~3 个"复审者"以便事先准备。每个复审者应该花 1~2 个小时复审工作产品、做笔记或者用其他方法熟悉这一工作。与此同时,复审主席也对工作产品进行复审,并制定复审会议的日程表,通常安排在第二天开会。



图 5-9 FTR 会议及其形式

复审会议由复审主席、所有复审者和生产者参加(图 5-9)。其中一个复审者作为"记录员",负责记录在复审过程中发现的所有重要问题。FTR 将从介绍会议日程开始,并由生产者做简单

的介绍。然后生产者“遍历”工作产品、做出解释,而复审者将根据各自的准备提出问题。当发现问题或错误时,记录员逐个加以记录。

在复审结束时,所有 FTR 的与会者必须做出以下决定中的一个。

(1) 工作产品可以不经修改而被接受。

(2) 由于严重错误而否决工作产品(错误改正后必须再次进行复审)。

(3) 暂时接受工作产品(发现必须改正的微小错误,但是不再需要进一步复审)。

做出决定之后,所有 FTR 与会者需要“签名”,以表示他们参加了此次 FTR 并且同意复审小组所做的决定。

在 FTR 期间,一名复审者(记录员)主动记录所有被提出的问题。在复审会议结束时,对这些问题进行小结,并生成一份“复审问题列表”。此外,还要完成一份简单的“复审总结报告”。复审总结报告将回答以下问题。

(1) 复审什么?

(2) 由谁复审?

(3) 发现了什么,结论是什么?

复审总结报告通常是一页纸大小(可能还有附件)。它是项目历史记录的一部分,有可能被分发给项目管理者和其他感兴趣的参与方。

本 章 小 结

保证软件的质量是软件工程的首要任务,好的软件质量是各级软件管理人员孜孜追求的最高梦想。但到底“什么是软件质量? 如何保证软件质量?”却是一个复杂的问题。为保证软件质量,必须严格定义软件质量的内涵,明确软件质量保证的策略,建立一组软件质量的保证活动,并付诸实施及不断改进和完善。只有这样,才能保证软件最终产品的质量。

软件质量保证(SQA)是一项计划的、系统的和规范性活动,是个全组织、多角色共同参与的、复杂的系统过程。ISO 9000-3 的核心思想是“将质量制作融入产品之中”。软件产品的质量取决于软件生存期所有阶段的活动。为把握产品的质量,必须使影响产品质量的全部因素在生产全过程中始终处于受控状态。

本章介绍了软件质量及其特性、软件质量模型、软件质量保证及其活动、软件配置管理等基本内容,为读者提供了软件质量管理的基本框架。软件质量保证向管理者提供对软件项目所采纳的过程和所开发的产品的质量信息,包括复查和审核软件产品及活动以验证它们是否符合试用的标准及规则。软件配置管理建立和维护在项目的整个生命周期内软件产品的完整性,包括软件配置项标识、版本控制、变更控制和对正式技术复审等活动,为整个软件生命周期内保持配置的完整性和可追踪性。

Dunn 和 Ullman 认为:“软件质量保证就是将质量保证的管理对象和设计原则映射到适用的软件工程管理和技术空间上。”质量保证的能力是成熟的工程学科的量度。当上述映射成功实现时,其结果就是成熟的软件工程。

思考与练习

1. 什么是质量？如何理解软件的质量？

2. 评价软件质量的常用模型有哪些？各有什么特点？

3. 什么是软件质量保证？软件质量保证活动包括哪些内容？

4. 软件配置管理的主要任务包括哪些内容？

5. 软件配置项包括哪些要素？如何标识软件配置项？

6. 如何理解基线？项目基线与项目里程碑有何区别？

7. 假定你是某个小项目的管理者，你将为项目定义什么基线？如何控制这些基线？

8. 版本控制的目的是什么？如何标识软件系统的版本号？

9. 设计一个项目数据库系统，它使软件工程师能够存储、交叉引用、跟踪、更新、修改所有重要的软件配置项。数据库将如何处理相同程序的不同版本？源代码的处理会与文档的处理有所不同吗？两个开发者将如何避免在相同时间内对相同的 SCI 进行不同的修改？

10. 研究现有的 SCM 工具，并描述它如何实现对于版本、变体及配置对象的控制。

11. 变更控制的目的是什么？请简述变更控制的流程。

12. 为什么要进行技术评审？如何进行正式技术复审？

13. 我们知道在一个多人参与的软件项目中可能出现的“通信成本”的例子。请给出一个反例，说明一群精通良好的软件工程实践并使用正式技术复审的工程师能如何提高项目组的生产率(当与个人生产率的总和相比时)。(提示：可以假定复审能够减少返工的工作量，而返工将占用个人工作时间的 20％～40％。)

第6章 软件风险管理

风险和收益总是结伴而行。一个项目之所以风险重重，是因为它把你带入未知的领域。逃避风险就等于举旗投降。全无风险的项目，它们的收益也几乎全无。

——(美)DeMarco《与熊共舞》

项目风险源于任何项目中都存在的不确定性。风险管理是软件项目管理的重要内容。软件风险管理就是通过主动而系统地对项目风险进行全过程的识别、分析和监控，最大限度地降低风险对软件开发的影响。软件风险管理的过程包括风险识别、风险分析、风险规划和风险监控等基本活动。风险管理的主要目标是预防风险。成功的项目管理一般都对项目风险进行了良好的管理，因此任何一个系统开发项目都应将风险管理作为软件项目管理的重要内容。

6.1 软件项目的风险管理

6.1.1 风险与项目风险

1. 生存主义者的启示

在美国有这样一群人，他们遍布美国各大城市，认为这个世界随时会发生毁灭性的自然灾害，会突然爆发世界大战，人类将要灭亡。于是他们时时刻刻做好准备，希望到时能够完全依赖自己的能力绝地求生。这群人被称为新生存主义者(New Survivalism)，生存狂或者生存偏执狂(图 6-1)。他们不仅担心战争，还担心恐怖主义、担心外星人袭击地球、担心小行星撞击地球，甚至还担心僵尸病毒。他们每天在为世界末日的到来做着积极的准备，一生都会苦苦锻炼身体、建造地下避难所、学习各种生存技能。尤其是 9.11 恐怖事件之后，他们的数量开始激增。美国几乎每个大城市都有生存主义者社团。

对于生存主义者，乐观主义认为他们是一群杞人忧天者，但从防范风险的角度来考虑，他们又是一批预防风险的先觉先行者，在潜在风险转变成灾难之前已做好了充分的准备。对大多数人来说，风险指的是在特定情况下某种结果的不确定性形式。某种事件可能会发生，如果发生，其结果对我们不利。它不是我们期望的结果。“风险”包含着对未来的疑虑与担心，风险一旦发生，其结果将使我们处于比现在更糟的境地。从被誉为“不沉之船”的“泰坦尼克”号的冰海折舟到 2008 年席卷全球的金融危机爆发，这些让人类付出惨痛教训的事例告诉并提醒我们，风险就存在于我们身边。要想安全生存就必须学会规避风险。



图 6-1 “新生存主义者”——手持的罐头上写着“有效期：世界末日之前”

阿里巴巴 CEO 的语录

阿里巴巴(Alibaba.com)创办人、首席执行官(CEO)马云也是互联网时代的典型生存主义者，作为一个互联网时代的幸存者，他成功地打开了“阿里巴巴”的财富之门。他有许多经典语录：

“一个公司在两种情况下最容易犯错误，第一是有太多的钱的时候，第二是面对太多的机会，一个 CEO 看到的不应该是机会，因为机会无处不在，一个 CEO 更应该看到灾难，并把灾难扼杀在摇篮里。”

“一个优秀的 CEO 和领导者，在给员工展示未来的时候不要光说美好，还要说到灾难。这样才能度过。”

这些话确实值得软件企业管理者和 CEO 深思。

2. 项目的一般风险

项目风险是一种不确定事件或状况，一旦发生，会对至少一个项目目标如时间、费用、范围或质量目标产生积极或消极影响[①]。在人类的大多数活动中，风险都以不同形式和程度出现。就项目风险而言，一般具有以下特征。

第一，风险存在的客观性和普遍性。作为损失发生的不确定性，风险是不以人的意志为转移并超越人们主观意识的客观存在，而且在项目的生命周期内，风险是无处不在、无时不在的。这些说明为什么虽然人类一直希望认识和控制风险，但直到现在也只能在有限的空间和时间内改变风险存在和发生的条件，降低其发生的频率，减少损失程度，而不能也不可能完全消除风险。

第二，某一具体风险发生的偶然性和大量风险发生的必然性。任何具体风险的发生都是诸多风险因素和其他因素共同作用的结果，是一种随机现象。个别风险事故的发生是偶然的、杂乱无章的，但通过对大量风险事故资料的观察和统计分析，发现其呈现出明显的运

① 项目管理知识体系指南.第 3 版.Project Management Institute，Four Campus Boulevard，Newtown Square，PA 19073-3299 USA，2004

动规律,这就使人们有可能用概率统计方法及其他现代风险分析方法去计算风险发生的概率和损失程度,同时也导致风险管理的迅猛发展。

第三,风险的可变性。这是指在项目的整个过程中,各种风险在质和量上的变化,随着项目的进行,有些风险可以得到控制,有些风险会发生并得到处理,同时在项目的每一阶段都可能产生新的风险。

第四,风险的多样性和多层次性。重大工程项目周期长、规模大、涉及范围广、风险因素数量多且种类繁杂,致使其在整个生命周期内面临的风险多种多样。而且大量风险因素之间的内在关系错综复杂,各风险因素之间与外界交叉的影响又使风险显示出多层次性,这是重大工程项目中风险的主要特点之一。

6.1.2 软件项目风险与管理

软件项目风险管理作为一门学科,出现于 20 世纪 80 年代末。经过 20 多年的发展,从理论、方法乃至实践上都取得了一定的进展。目前,随着软件工程技术的进步和软件企业的不断成熟,其研究已成为软件工程和项目管理中的热点问题之一。

长期以来,人们会对一座价值数百上千万的桥梁或建筑物的垮塌给予极大的关注,认为这是不可饶恕的责任事故,而对一项价值百万的软件项目夭折则可归为高科技学费。所以多年来软件项目建设者们非常幸运,公众容易接受软件项目失败的现实,容易原谅导致失败的各种原因。造成这种现象的部分原因是软件项目的复杂性和不可视性,但也从另一个侧面说明了软件科学的不成熟性。实际上,由于软件技术的飞速发展和软件系统在社会生产各方面的广泛应用,各种风险因素及风险发生的可能大大增加,并且扩大了风险事件造成的损失规模,这就对软件项目提出了更高的管理要求,使风险管理成为项目管理的重要过程。软件项目的潜在风险,甚至生命攸关的危险照样存在,且还会因认识的不足而造成更大危害。

软件项目风险是指软件开发过程中存在大量的需求、技术、人员、过程、组织等方面的不确定性,可能导致软件产品/服务的功能不能满足要求、费用超出预算、进度延迟或项目被迫取消等所不期望的后果[①]。软件风险管理则是对可能导致上述不利影响的风险因素进行评估和控制,将风险降至管理者可以接受的范围内。

对软件项目风险概念的理解源于其他工程项目风险管理。项目管理领域中新的突破,往往能给软件项目的风险管理提供有益的参考,但如何应用于软件项目风险管理并发挥作用,还是目前研究的热点问题之一。软件项目风险管理可借用多个学科的理论、方法和工具,但主要内容集中在工程领域。软件工程领域理论的深入发展也需要开展风险管理的研究,并通过借鉴工程项目的管理办法来解决软件项目中出现的风险问题。

人类面对复杂系统难免会出现判断失误,特别是当复杂程度超出了人类当前的认知能力时。系统的复杂程度和所引起的风险如图 6-2 所示,呈现出明显的非线性关系。随着风险程度的升高,原本可以靠个人能力处理风险的经验已不再适用,当风险继续增大时,则必须在某些方面采用专门的方法,以及采用专门的工具。

---

① 徐如志,等.软件风险管理及优化控制.计算机工程,2005,31(9):73-75

软件项目迫切需要加强风险管理，这主要表现在以下两个方面。

第一，风险管理是软件工程发展的必然要求。与一般项目相比，软件项目具有更大的不确定性。在项目的实施中，各任务的执行都存在与计划偏离的风险，这些风险就像多米诺骨牌一样，可以在软件项目的任务、过程或小组之间进行传递和累积，并最终影响整个项目的风险水平(图 6-3)。“事实上，即使在最成功的软件项目的每一个阶段，都有大量的有用要素是未知的”①，而防止项目失控最好的方法就是对软件项目进行风险管理。

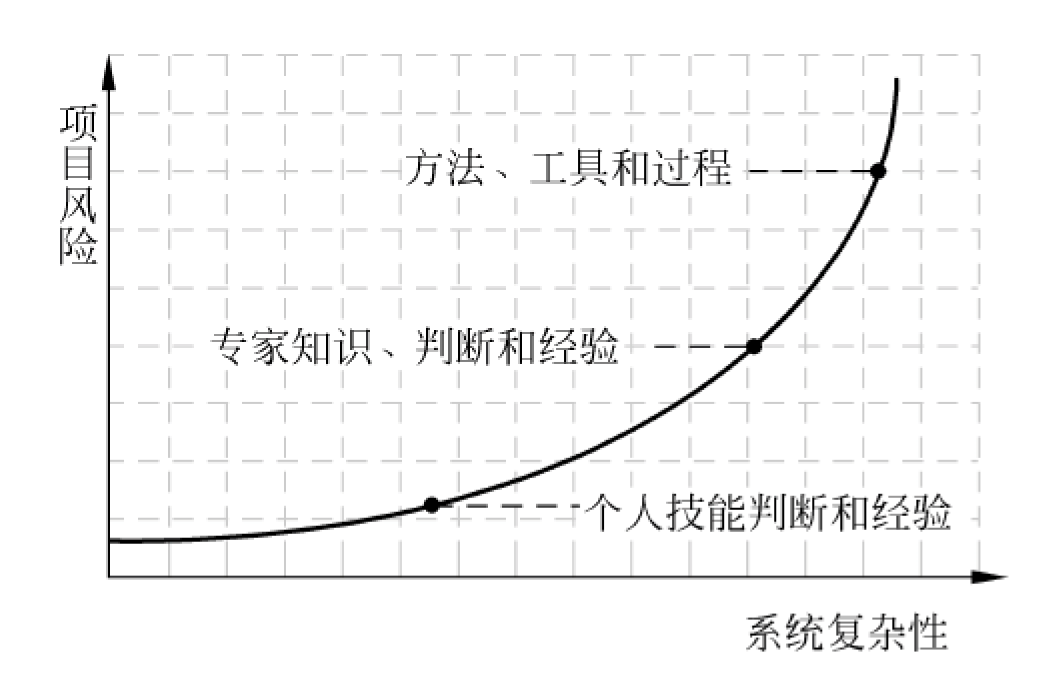


图 6-2 复杂性、风险和解决办法之间的关系

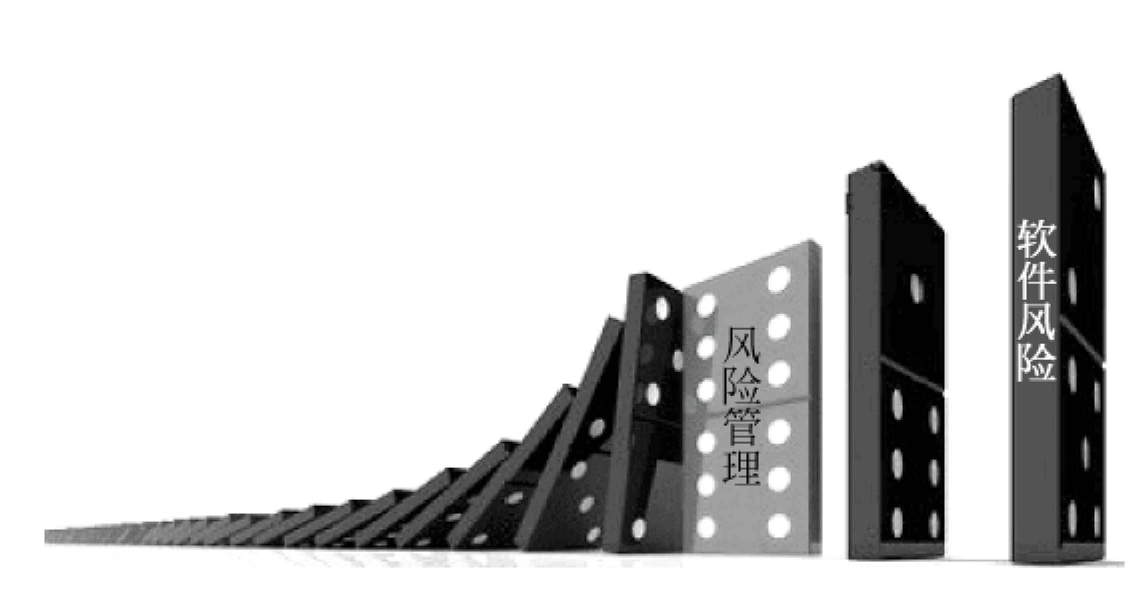


图 6-3 风险在项目过程中就像多米诺骨牌一样可以传递和累积

第二，风险管理已经成为软件业界减少意外发生的一个最佳实践。虽然我们不能很确切地预测未来发生的事情，但是可以使用结构化的风险管理实践活动尽早地发现还处于萌芽状态的各种陷阱。风险管理方法的不断改进及其在许多典型项目中的成功应用，使软件项目团队自觉地引入了风险管理机制。因此风险管理被认为是管理软件项目特别是管理大型软件项目的重要的方面。正如软件风险管理的奠基人之一 Charette 所认为的那样：“大型软件项目的管理就是风险管理。”

在软件业，学术界和企业界都越来越强烈地相信，没有一个独立的方法、技术、工具或过程能够解决软件项目失控问题，驾驭项目失控最好的方法是从开始就管理项目的风险。KPMG (毕马威)在 1995 报告中列举的项目失控企业，55%的失控项目没有实行过任何风险管理，而在 38%实行了风险管理(有些调查者不知道是否实行了风险管理)的项目中，有 50%的项目在启动之后没有使用风险发现(Risk Finding)，缺少风险管理可能会导致项目失控。管理项目风险的好处是明显的，Boehm(1989)认为，风险管理之所以重要，是因为它使人们脱离灾难，避免返工，并促使软件项目取得双赢的局面。

大量实例说明，许多信息系统失败于不恰当的项目风险管理。研究表明②，约有 1/3 的软件项目，由于未能及时预见和有效控制软件风险，导致软件过程失控而被迫取消；另外 2/3 的软件项目，由于同样的原因，实际开发时间平均是计划的两倍。IEEE 的研究认为③：软件系统中 50%～70%的风险可以被检测到，90%的风险可以避免，风险管理的杠杆作用，

---

① Jim McCarthy. Dynamics of Software Development. Redmond，Washington：Microsoft Press，1995：99

② Brown W. AntiPatterns. Refactoring Software，Architectures，and Projects in Crisis. New York，Wiley Computer Publishing，1998

③ Lister Tim. Interview with Tim Lister[J]. IEEE Software，1997：3～4

一般的投资回报率是700%～2000%，风险评估中耗时、耗力的工作可以借助软件包得以减轻。这些数据表明，软件项目的风险是固有，也是可以被认识、被预测，被驾驭的。

风险扶梯[①]

在今天的环境下如何对待风险？风险管理专家Bob Charette提出了一种全新的思路。他建议读者把自己的企业和竞争对手看成是处在一组向下的自动扶梯上，在这个设想的扶梯世界里，新的竞争者是从扶梯中间进入的。你必须奋力跑到扶梯的顶端，你的竞争对手也在他们各自的扶梯上做着同样的事。也就是说，如果你落后，新到的竞争者一定会出现在你的前面。扶梯运转得越快，你就必须跑得越快，才能让自己保持在原来的位置。如果你停下脚步，哪怕只是一小会儿，就会落于下风。当然，如果你停得太久，就会掉到扶梯的底端，丧失竞争的资格。

6.1.3 软件风险的定义

虽然对于软件风险的严格定义还存在很多争议，但在风险中包含了两个特性，在这一点上是已达成了共识的。

如最早研究软件项目风险管理的美国国防部，把风险定义为：在预定成本、工期和技术约束下，可能无法达到全面计划目标的度量指标，它包含以下两部分：

（1）无法达到具体结果的概率（或可能性）。

（2）达不到那些结果的后果（或影响）。

Boehm等将这两部分归结为“风险揭露”（Risk Exposure，RE）[②]，用公式可以表示为：

$$RE = P \cdot C \tag{6-1}$$

其中：RE 表示风险揭露；P 表示风险发生的不确定性（用概率表示事件发生的可能性）；C 表示风险产生时带来的损失程度（例如项目成本）。

通过风险揭露来量化风险为所有已知风险提供了相对的参考。例如：

对某软件产品测试能提高产品的可靠性，其正确性概率为0.8，但测试可能会增加成本耗费15 000元，则其风险揭露＝0.8×150 000＝12 000元。

以上的风险揭露公式未指明其主体，即是什么造成的不利影响，所以有些文献又将风险主体表示为“场景”。软件项目风险管理的另一位创始人Charette将风险定义为一个三元组[③]，即：“发生的有害事件是什么？发生的可能性有多大？如果发生引致的后果如何？”其形式化表示为：

$$R_{risk} = \{\langle s_i, l_i, x_i \rangle\}_c \tag{6-2}$$

其中：R 代表风险；s_i 为第 i 个有害事件；l_i 代表第 i 个有害事件发生的概率；x_i 表示第 i 个事件的结果，是一种损失指标；脚标 c 表示这个集合是一个完备集。这3者构成了评估风险的基础。据此可以认为，风险不是一个数字，也不是一条曲线或是一个向量，而应该是一个三元组的完备集，整个集合才是全部风险。

1997年，在风险分析学会的大会报告中，Kaplan进一步完善了这种完备集风险的定

---

① 迪马可.与熊共舞——软件项目风险管理.熊节，等译.北京：清华大学出版社，2004

② Boehm B. W.. Software risk management. Piscataway：IEEE Computer Society Press，1989

③ Charette R.. Software engineering risk analysis and management. New York：McGraw-Hill，1989

义。他从多年来学术界对概率定义的争论出发，指出可能性有 3 种表达：频率、概率和频率的概率，其中频率的概率是最有说服力，最适用的。

$$R_{risk} = \{\langle s_i, p_i(\varphi_i), p_i(x_i)\rangle\}_c \tag{6-3}$$

其中：s_i 依然为第 i 个有害事件；φ_i 为第 i 个事件发生的频率，$p_i(\varphi_i)$代表第 i 个事件发生频率为 φ_i 的概率；$p_i(x_i)$指第 i 个事件的结果为 φ_i 的概率，它是一个向量，与事件不独立。显然，该完备集风险的定义在量化上是一个进步。

经过对风险的形式化定义，使人们对软件风险的概念逐渐加深，为理论研究奠定了基础。

6.1.4 软件风险的类型

在进行风险分析时，重要的是量化不确定性的程度及与每个风险相关的损失的程度。为了实现这点，必须考虑不同类型的风险。一般认为软件风险有 3 种类型，即项目风险、技术风险与商业风险(图 6-4)。



图 6-4 软件风险的类型

项目风险是指潜在的预算、进度、人力(工作人员及组织)、资源、客户及需求等方面的问题以及它们对软件项目的影响。项目风险威胁到项目计划。也就是说，如果项目风险变成现实，有可能拖延项目的进度，且增加项目的成本。项目风险的因素还包括项目的复杂性、规模、结构的不确定性。

技术风险是指潜在的设计、实现、接口、验证和维护等方面的问题。此外，规约的二义性、技术的不确定性、陈旧的技术，以及“过于先进的”技术也是风险因素。技术风险威胁到要开发软件的质量及交付时间。如果技术风险变成现实，则开发工作可能变得很困难或根本不可能。

软件产品投入的收益水平是受商业驱动的，商业风险常常会危害项目或产品，威胁到要开发软件的生存能力。5 个主要的商业风险是：①开发了一个没有人真正需要的优秀产品或系统(市场风险)；②开发的产品不再符合公司的整体商业策略(策略风险)；③建造了一个销售部门不知道如何去卖的产品；④由于重点的转移或人员的变动而失去了高级管理层的支持(管理风险)；⑤没有得到预算或人力上的保证(预算风险)。

事实上，正是由于风险的复杂性，分类的依据或方法、方面还有很多，例如 Boehm 把风险分为一般风险和项目特定风险，前者可由标准的开发计划技术处理，后者要由特定项目风险管理计划处理。SEI 把风险分为两大类：管理和技术。管理包括项目风险和管理过程的风险，技术包括产品风险和技术过程的风险。应该认识到，任何单一的分类方法均无法全面刻画软件项目风险的本质属性，并且几种分类的简单堆砌也无法有效提高风险辨识、评估和控制的效率，对风险的认识必须上升到项目系统性和整体性的层面上。

风险与投资

当风险处于凭直觉管理时代时，风险辨识主要靠的是管理者个人的经验、能力和对风险的感受，项目成败很大程度上受制于项目的具体执行者（图 6-5）。

面对各种商业风险和投资风险，联想总裁柳传志曾经直言不讳地说："没钱赚的事不能干；有钱赚但是投不起钱的事不能干；有钱赚也投得起钱但是没有可靠的人去做，这样的事也不能干。"



图 6-5　风险与投资

6.2　软件风险管理的体系框架

6.2.1　常见风险管理过程框架

人们通常以过程的形式对大多数特定的风险管理进行描述，这些过程又以各种各样的方式被进一步分解为阶段或活动。下面是一些常用的和有影响的风险管理过程框架。

1. Boehm 体系

在软件风险体系方面，Boehm 在他的奠基性著作《软件风险管理》中[①]，认为软件风险管理指的是"试图以一种可行的原则和实践，规范化地控制影响项目成功的风险"，其目的是"辨识、描述和消除风险因素，以免它们威胁软件的成功运作"。Boehm 认为依照风险源清单（Risk Checklist），可发现项目中大多数严重的风险因素，给出了前十大风险源序列。之后又于 1991 年，结合对一些经验丰富的软件项目管理人员的调查，提出了顶级十大风险源清单[②]，奠定了该领域的理论基础。在他的著作与论文中，Boehm 比较详细地对软件开发中的风险进行了论述，并提出软件风险管理的方法。他把风险管理活动分成风险估计和风险控制两大阶段，风险估计阶段包括风险辨识、风险分析和风险排序 3 项活动，风险控制阶段包括风险管理计划、风险处理和风险监督 3 项活动。

Boehm 为每项活动都提供了不少的相关实现技术。例如，风险识别中给出了 10 种软件风险因素清单，同时还推荐了各因素的相关处理意见及方法。从该清单出发，管理人员和技术人员能够进一步细化风险因素，并给予评估和化解。

2. Charette 体系

Charette 设计的风险管理体系分为两大阶段，分别为分析阶段和管理阶段，每个阶段内含 3 个过程，如风险分析阶段包括风险标识、风险估计、风险评估 3 个过程。这是一个相互重叠和循环的模型。Charette 同时为各个过程提供了相应的战略思路、方法模型和

---

① Boehm B. W.. Software Risk Management. IEEE Computer Society Press Washington D. C，1989

② Boehm B. W.. Software Risk Management：principles and practices. IEEE Software，1991，8(1)：32～41

技术手段。特别在风险的辨识和估计过程中，其中大多数是运筹学、系统科学中的模型应用。

3. 项目持续风险管理

项目持续风险管理（Continuous Risk Management，CRM ）理论最初由美国卡耐基·梅隆大学软件工程研究所(Software Engineering Institute，SEI)在 20 世纪 90 年代中期提出，后来被美国航空航天局(National Aeronautics and Space Administration，NASA)等部门应用，CRM 已成为 NASA 项目风险管理培训的核心课程。现在该理论的应用已扩展到适用于硬件项目和其他复杂系统或组织的管理领域。

CRM 是一种对项目风险进行动态管理的理论，按照风险管理的整个过程，将风险管理的主要活动，分为六大过程模块，即风险识别、风险分析、风险规划、风险跟踪、风险控制和风险文档记录(图 6-6)。其中风险识别、风险分析、风险规划、风险跟踪和风险控制模块在风险过程上首尾相连，风险文档记录模块贯穿以上五大模块，共同构成风险管理的基本内容。风险文档记录是 CRM 的关键步骤模块，记录整理风险管理过程，风险管理信息，尤其是项目内外出现的新情况，产生的新风险，将这些新的项目信息和新的风险通过正式的风险记录报告或非正式的信息传递方式告知给 CRM 的其他五大模块。

CRM 模型要求在项目生命期的所有阶段都关注风险识别和管理，框架显示了应用 CRM 的基础活动及其之间的交互关系，强调了这是一个在项目开发过程中反复持续进行的活动序列。每个风险因素一般都需要按顺序经过这些活动，但是对不同风险因素开展的不同活动可以是并发的或者交替的。

4. Microsoft 的风险管理体系

风险管理是 Microsoft 解决方案框架(MSF)的核心原则之一。MSF 认为，变化和随之产生的不确定性是软件生命周期与生俱来的特征。MSF 风险管理原则提倡使用一种主动的方法来处理这种不确定性，连续地评估风险，并在生命周期中使用它们来影响最终决策。这一过程瞄准成功的、持续的风险管理，通过使用如图 6-7 所示的 6 个阶段来描述过程、原则以及指导。①

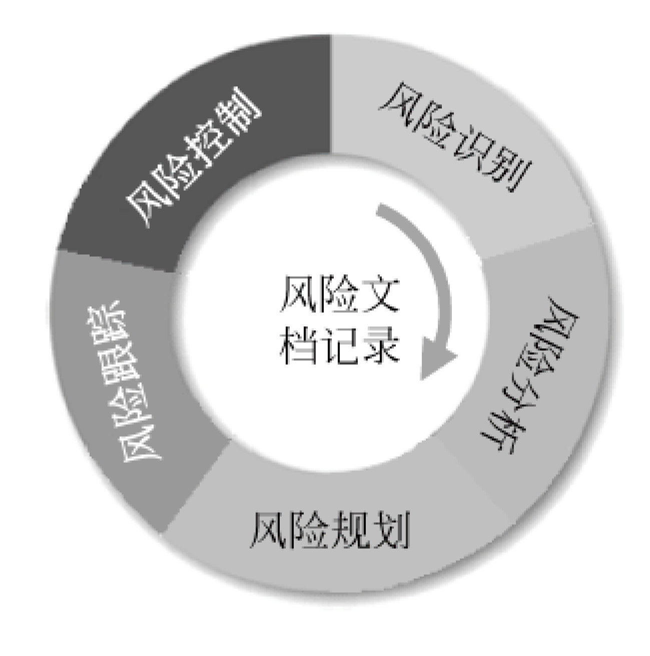


图 6-6　CRM 风险管理活动

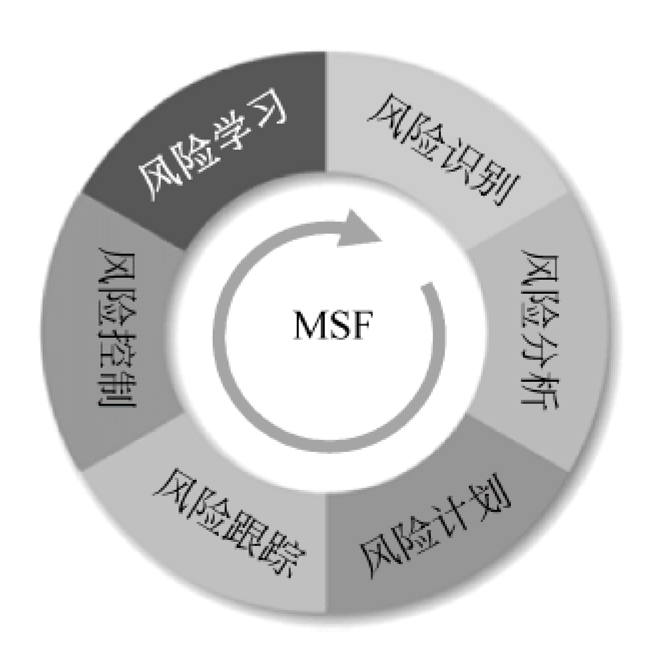


图 6-7　MSF 的风险管理活动

① Microsoft 解决方案框架：风险管理原则 v. 1. 1，http://www.microsoft.com/china/technet/itsolutions

MSF 的风险管理思想是,风险管理必须是主动的,它是正式和系统的过程,风险应被持续评估、监控、管理,直到被解决或问题被处理。该模型最大的特点是将学习活动融入风险管理,强调了学习以前项目经验的重要性。

MSF 的风险管理原则是:①持续的评估;②培养开放的沟通环境:所有组成员应参与风险识别与分析,领导者应鼓励建立没有责备的文化;③从经验中学习:学习可以大大降低不确定性,强调组织级或企业级的从项目结果中学习的重要性;④责任分担:组中任何成员都有义务进行风险管理。

4 种风险管理框架如表 6-1 所示。

表 6-1　典型的风险管理框架

Boehm 风险管理体系	Charette 风险管理体系	SEI 风险管理体系	Microsoft 风险管理体系
风险评估	**风险分析**	风险识别	风险识别
风险识别	风险标识	风险分析	风险分析与分级
风险分析	风险估计	风险规划	风险计划和调度
风险排序	风险评估	风险跟踪	风险跟踪和报告
风险控制	**风险管理**	风险控制	风险控制
风险管理计划	风险计划	风险文档记录	风险学习
风险处理	风险控制		
风险监控	风险监控		

6.2.2　软件风险管理的一般过程

综合以上几个软件风险体系,可以看出它们之间的共性,软件风险管理实际上就是贯穿在项目开发过程中的一系列管理步骤。过程就是基于一定输入,采用相关工具和技术,产生一定输出的活动集合。这里,我们将风险管理过程划分成 4 个相关阶段:风险识别、风险估计、风险规划和风险监控(图 6-8)。

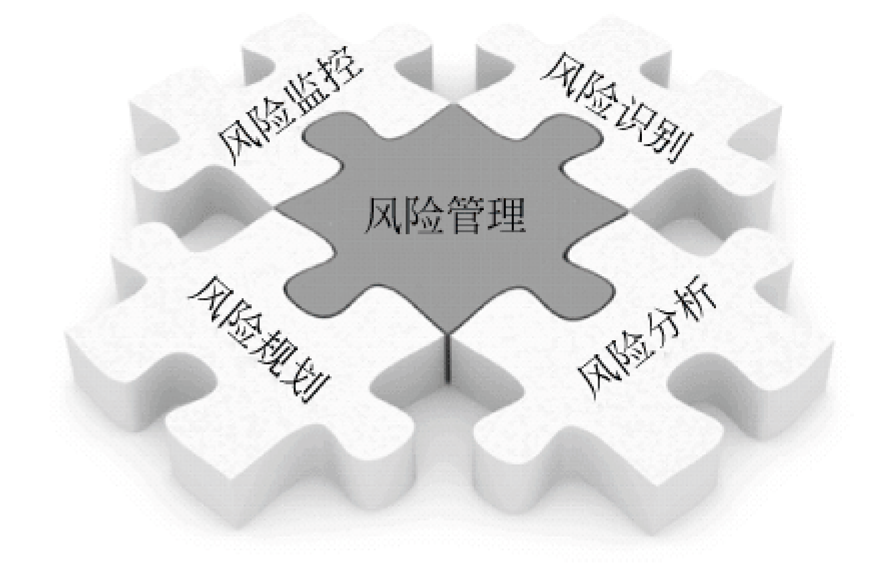


图 6-8　软件风险管理过程

(1) 风险识别。主要是确定风险事件及其来源。

(2) 风险分析。对已识别的风险要进行分析和评估,风险分析的主要任务是确定风险发生的概率与后果。

(3) 风险规划。制定风险响应的措施和实施步骤,按照风险的大小和性质,制定相应的措施去应对和响应风险,包括风险接受、风险转移等。

(4) 风险监控。包括对风险发生的监督和对风险管理的监督,前者是对已识别的风险源进行监视和控制,后者是在项目实施过程中监督人们认真执行风险管理的组织和技术措施。

6.3 风险识别

所有对项目风险管理过程的描述都强调，在过程启动的初始阶段要对风险的来源进行识别。所以，风险识别(Risk Identification)是任何风险管理活动的起点和基础。

6.3.1 风险识别过程

风险识别是项目管理者识别风险来源、确定风险发生条件、描述风险特征并评价风险影响的过程。项目规划中的项目目标、任务、范围、进度计划、成本预算、资源计划、采购计划及利益相关方对项目的期望值等都是项目风险识别的主要依据。通过风险识别，可以将那些可能给项目带来危害和机遇的风险因素识别出来。风险识别是制定风险分析和风险规划的依据。

根据项目管理的知识体系(PMBOK)，风险识别过程定义如图 6-9 所示的 IDEF0 图。

IDEF0 是活动模型的缩写，来源于结构化分析与设计技术的一套标准。在如图 6-10 所示的过程图中，IDEF0 图通过控制、输入、输出和机制描述了顶级过程。控制(Control)决定过程何时和如何执行，位于图的上方；输入(Input)实行或完成特定活动所需的资源，位于图的左侧；输出(Output)是过程转变的结果，位于图的右侧；机制(Mechanisms) 是完成活动所需的工具与方法，包括人员、设施及装备，位于图的底部。箭头用于连接系统中各活动，通常由名词描述。

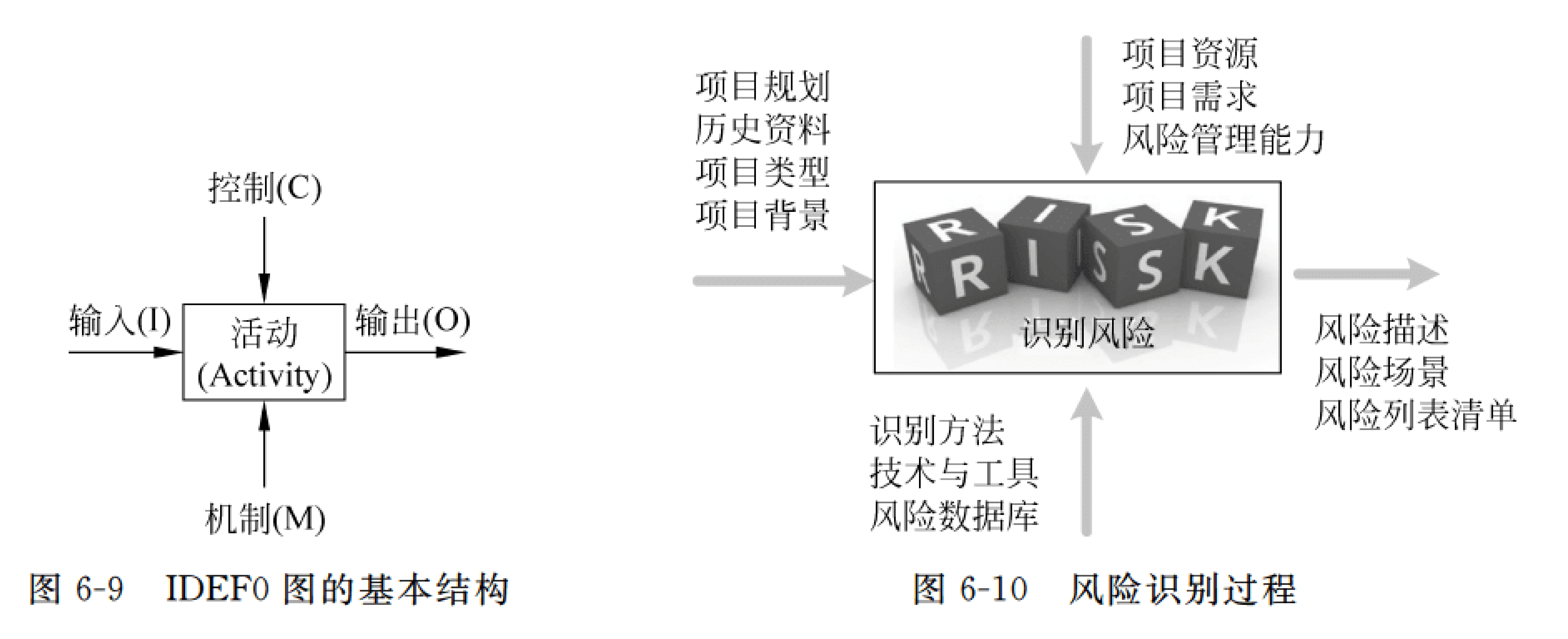


图 6-9 IDEF0 图的基本结构　　图 6-10 风险识别过程

1. 过程控制

项目资源、项目需求和风险管理能力用来调节风险识别过程。项目资源用成本、时间和人员限制风险识别的范围；合同化的需求能直接说明风险评估的需求；风险管理的核心能力就是风险管理能力，是项目管理的核心价值所在，也是软件风险管理的重点。

2. 过程输入

风险识别过程的输入包括项目规划中的项目目标、任务、范围、进度计划、费用计划、资源计划、利益相关方对项目的期望值等，都是项目风险识别的依据。

项目风险识别的另一个重要输入或依据就是历史资料，即从本项目或其他相关项目的档案文件中、从公共信息渠道中获取对本项目有借鉴作用的风险信息。以前做过的、同本项目类似的项目及其经验教训对于识别本项目的风险具有借鉴作用。

就项目类型而言，一般来说，普通项目或重复率较高项目的风险程度比较低，技术含量高或复杂性强的项目的风险程度比较高。项目类型还与项目所在行业及应用领域有密切的关系，不同的项目类型可能产生风险的风险源有所不同。例如，在金融领域，对安全性要求就很高。对于控制系统，对系统的可靠性有特别的要求。掌握了各类项目的特征规律，也就掌握了风险识别的钥匙。

项目背景提供了与项目相关的直接与间接信息（事件、条件、约束、假设、环境、诱发因素和相关问题）。例如，项目必然处于一定的环境之中，受到内外许多因素的制约，其中有些因素是项目活动主体无法控制的。这些构成了项目的制约因素，是项目管理人员所不能控制的，这些制约因素中隐藏着风险。另外，项目在规划时，一般是在若干假设、前提条件下提出或估算出来的。这些前提和假设在项目实施期间可能成立，也可能不成立。因此，项目的前提和假设之中隐藏着风险。

3. 过程输出

风险识别过程的输出是风险描述和与之相关的风险场景。风险描述是用标准的表示法对风险进行简要说明，如项目风险来源清单、项目风险征兆、项目风险类别，以及项目风险发生的可能性、将会产生的后果和影响等。风险场景提供了与风险背景相关的间接信息，如对事件、条件、约束、假定、环境、有影响的因素和相关问题等的场景描述。

风险识别过程的一个重要输出是提供一个风险清单，清单上列举出在任何时候可能碰到的风险。

4. 过程机制

风险识别方法、分析工具和风险数据库是风险识别过程的机制。机制可以是方法、技巧、工具或为过程活动提供结构的其他手段。风险数据库是一个已知风险和相关信息的仓库，它将风险输入计算机，并分配一个标识码给这个风险，同时维持所有已识别风险的历史记录。

什么是 IDEF 图

IDEF 是 ICAM DEFinition method 的缩写，是美国空军在 20 世纪 70 年代末 80 年代初 ICAM（Integrated Computer Aided Manufacturing）工程在结构化分析和设计方法基础上发展的一套系统分析和设计方法。最初的 IDEF 方法有 3 种：功能建模（IDEF0）、信息建模（IDEF1）、动态建模（IDEF2），后来，随着信息系统的相继开发，又开发出了 IDEF 族方法。IDEF0 用来描述对于企业具有重要性的各个过程（活动）。它以图形表示完成一项活动所需要的具体步骤、操作、数据要素以及各项具体活动之间的联系方式。有关 IDEF0 图的较为详细的介绍可参考本书附录 A 中相关内容。

6.3.2 风险识别的方法与工具

项目风险识别过程活动的基本任务是将项目的不确定性转变为可理解的风险描述。在项目风险识别过程中一般要借助一些技术和工具，这样识别风险的效率高而且操作规范，不容易产生遗漏。在具体应用过程中要结合软件项目的具体情况，组合起来应用这些工具。

1. 检查表

检查表是管理中用来记录和整理数据的常用工具。用它进行风险识别时，将项目可能

发生的许多潜在风险列于一个表中，供管理人员检查核对，用来判别某项目是否存在表中所列或类似的风险。检查表中所列都是历史上类似项目曾发生过的风险，是项目风险管理经验的结晶，对项目管理人员具有开阔思路、启发联想、抛砖引玉的作用。表 6-2 给出一个按风险类型分类的风险检查表的参考模板。

表 6-2　风险检查表参考模板

风险类型	序号	风险检查条目
项目风险	1	对项目的规模估算是否比较正确
	2	项目资源能够得到保证吗
	3	项目进度安排是否合理？有合适的缓冲时间吗
商业风险	1	政府或者其他机构对本项目的开发有限制吗
	2	本项目的市场前景如何？有不可预测的市场风险吗
	3	客户的信誉好吗？例如按客户的需求开发了产品，但是客户可能不购买
技术风险	1	需求开发人员是否真正获取或理解用户需求
	2	开发人员是否有开发相似产品的经验？是否已经掌握了本项目的关键技术
	3	开发小组是否采用比较有效的分析、设计、编程、测试工具

2. 图解技术

(1) 因果关系分析法

因果关系分析法用于揭示影响及其原因之间的联系，以便追根溯源，找出风险的根本原因。其目的是通过确定问题的根本原因来阻止问题的发生。因果分析图(又称鱼骨图)是在因果关系分析法中常用的一个重要工具。鱼骨图分析方法首先确定问题特性，然后列出主要原因以及分析次要原因，最后统计各种原因的概率，得出影响该问题的主要因素。

因果分析图一般是根据风险列表等方法分析风险的存在，在假设风险存在的基础上画出鱼骨图，确定风险起因的方法。图 6-11 中以软件开发的需求风险为例，用鱼骨图来分析导致该风险的各种主要原因及次要原因。

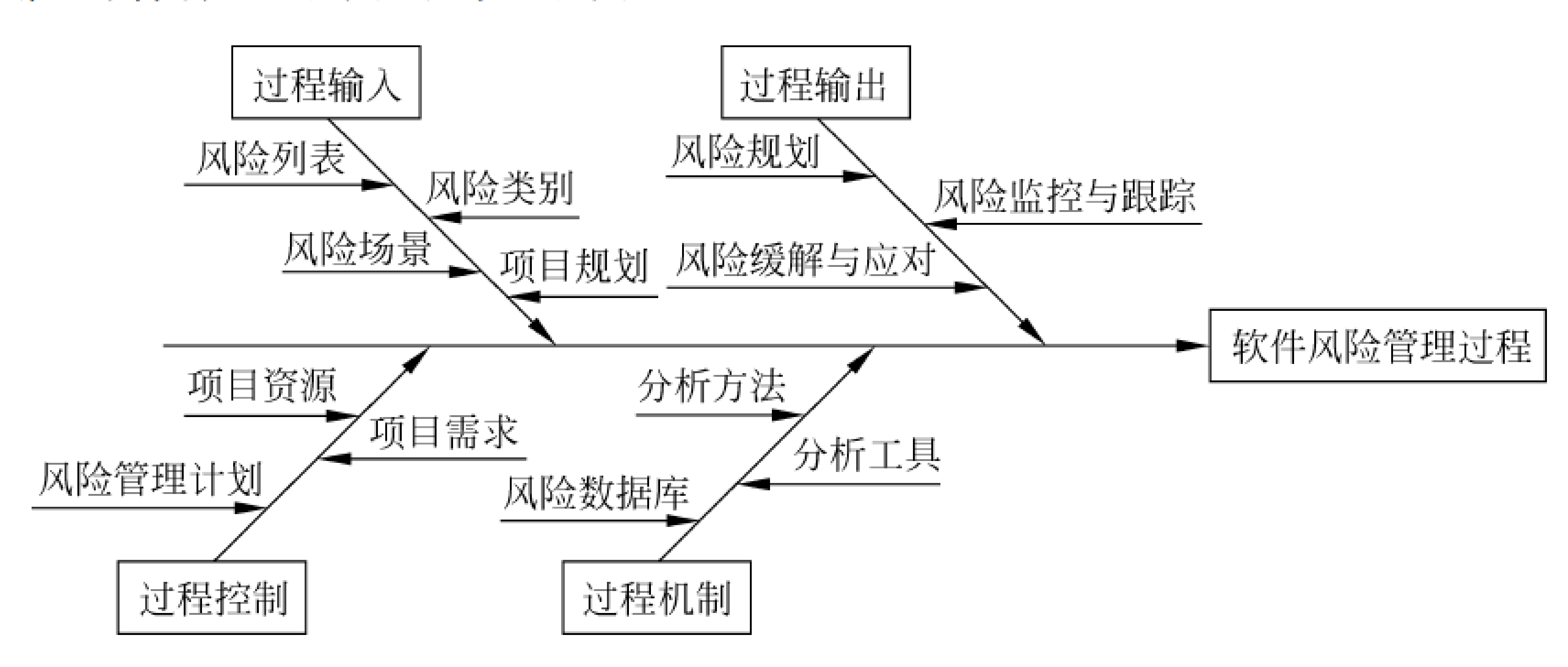


图 6-11　软件风险管理过程的因果分析图

(2) 流程图

流程图是又一种项目风险识别时常用的工具。流程图可以帮助项目识别人员分析和了解项目风险所处的具体项目环节、项目各个环节之间存在的风险以及项目风险的起因和影响。通过对项目流程的分析，可以发现和识别项目风险可能发生在项目的哪个环节或哪个

地方,以及项目流程中各个环节对风险影响的大小。

3. 头脑风暴法

头脑风暴(Brain Storming)是指有计划的集体想象活动:群策群力,发挥集体的创造性,使新的思想得以浮现。

头脑风暴法遵循五大原则:①禁止评论他人的构想;②最狂妄的想象是最受欢迎的;③重量不重质,即为了探求最大量的灵感,任何一种构想都可被接纳;④鼓励利用别人的灵感加以想象、变化、组合等以激发更多更新的灵感;⑤不准参加者私下交流,通过头脑风暴法收集项目风险,然后风险管理人员将会议结果进行分类整理,最后填写风险管理表,提交风险分析。

4. 情景分析法

众所周知,在拥有大量历史数据,而且关键变量间的关系在未来保持不变时,传统的统计预测方法比较有效。但在动荡多变和错综复杂的环境下,统计预测方法因其基于关键变量间的历史联系的假设跟实际情况不符而很难奏效。与之相反,情景分析法因其明确地聚焦于长期计划的假设而显得格外突出[①]。特别是在"复杂的"问题中,事件因素和相关假设的相关性并不拘泥于某种形式。假如某个重要的参数没有被定义,有可能导致整个研究陷入泥潭。

情景分析(Scenario Analysis)中所指的"情景"是指对事物所有可能的未来发展态势的描述,内容包括对各种态势基本特征的定性和定量描述。它是根据发展趋势的多样性,通过对系统内外相关问题的系统分析,设计出多种可能的未来前景,然后用类似于撰写电影剧本的手法,对系统发展态势做出自始至终的情景和画面的描述。当一个项目持续的时间较长时,在制定战略规划时都使用该方法。一些国家政府也采用了该方法,如历史上南非白人政府的种族隔离制度的和平变革,就是利用该方法推导了各种选择可能的结果之后做出的选择。



图 6-12 情景分析步骤

现在大多数国际组织和公司更常用的是斯坦福研究院(Stanford Research Institute,SRI)拟定的6项步骤(图6-12)[②]。

(1) 明确决策焦点。明确所要决策的内容项目,以凝聚情景发展的焦点。

(2) 识别关键因素。确认所有影响决策成功的关键因素,即直接影响决策的外在环境因素。

(3) 分析外在驱动力量。确认重要的外在驱动力量,包括政治、经济、社会、技术各层面,以决定关键决策因素的未来状态。

---

① 曾忠禄,张冬梅.不确定环境下解读未来的方法:情景分析法[J].Journal of Information 2005,5:14

② 岳珍,赖茂生.国外"情景分析"方法的进展[J].情报杂志,2006(7):59~61

(4) 选择不确定的轴向。将驱动力量以冲击水平程度与不确定程度按高、中、低加以归类。在属于高冲击水平、高不确定的驱动力量群组中，选出 2～3 个相关构面，称之为不确定轴面，以作为情景内容的主体构架，进而发展出情景逻辑。

(5) 发展情景逻辑。选定 2～3 个情景，这些情景包括所有的焦点。针对各个情景进行各细节的描绘，把故事梗概完善为剧本。

(6) 分析情景内容。可以通过角色试演的方法来检验情景的一致性，这些角色包括多个方面。通过这一步骤，管理者可以根据自己的观点进行辩论并达成一致意见，更重要的是，管理者可以看到未来环境里各角色可能做出的反应，最后认定各情景在管理决策上的含义。

6.4 风险分析

风险分析是评估已识别出风险的影响和可能性的过程。风险分析可以选择定性分析或定量分析方法，进一步确定已识别的风险对项目目标的影响，并根据其影响对风险进行排序，确定项目的关键风险项，并指导接下来的风险应对计划的制定。

6.4.1 风险分析过程

风险分析过程的活动是将风险识别的输出(重要的一项是风险列表)转变为优先顺序排列的风险列表所需的任务。根据 PMBOK 风险处理框架，风险分析过程定义参见如图 6-13 所示的 IDEF0 图。



图 6-13 风险分析过程

1. 过程控制

项目资源、项目需求和风险管理计划调节风险分析过程，与风险识别过程类似。

2. 过程输入

在进行风险分析时，一个重要的任务是量化每个风险的不确定性和可能造成损失的程度。风险列表清单给项目管理提供了一种简单的风险预测技术。

项目状态是项目进展状况的标识，风险的不确定性常常与项目所处的生命周期阶段有关。在项目初期，项目风险症状往往表现得不明显，随着项目的进程，项目风险及发现风险的可能性会增加。项目状态可与基线和交付物作为评价依据。

3. 过程输出

按优先等级排列的风险列表及其趋势分析是风险估计过程的输出。一个按优先等级排列的风险列表是一个详细的项目风险目录,其中包含了所有已识别风险的相对排序及其影响分析。项目管理者对排序进行研究,并划分重要和次重要的风险,对重要的风险要进行管理,对次重要的风险再进行一次评估并排序。

4. 过程机制

风险分析方法、分析工具和风险数据库是风险估计过程的机制。机制可以是方法、技巧、工具或为过程活动提供结构的其他手段。风险分析工具可有效地帮助人们管理不确定性因素。通过风险分析,可制定有效的决策。用自动工具进行风险分析,即可保存、组织和处理数据,使其变为有意义的资料。

6.4.2 风险分析的技术与工具

1. 风险列表

风险列表给项目管理者提供了一种简单的风险预测技术(样本如表 6-3 所示)。项目组一开始要在表中的第一列列出所有风险可能,这些可以利用前面所述的风险检查条目来完成;在第二列对风险进行分类;风险发生概率放在第三列。每个风险的概率值可以由项目组成员个别估算,然后将这些值平均,得到一个有代表性的概率值。

一旦完成风险表的前 4 列内容,就要根据概率及影响来进行排序。高概率、高影响的风险放在表的上方,依次类推。这就完成了第一次风险排序。项目管理者研究已经排序的表,并定义一条终止线。该终止线(表中某一点上的一条水平线)表示:只有在那些线上的风险才会得到进一步的关注,线之下的风险则需要再评估以完成第二次排序。

从管理的角度来考虑,风险的影响及概率是起着不同作用的,一个具有高影响且发生概率很低的风险因素不应花太多时间,而高影响且发生率从中到高的风险以及低影响且高概率的风险,应该首先列入管理考虑之中。

表 6-3　风险列表样本

风　险	类　别	概　率	影　响	RMMM
规模估算可能非常低	项目风险	60%	2	
用户数量大大超出计划	项目风险	30%	3	
复用程度低于计划	技术风险	70%	2	
最终用户抵制该计划	商业风险	40%	3	
交付期限将被紧缩	商业风险	50%	2	
资金将会流失	商业风险	40%	1	
用户将改变需求	项目风险	80%	2	
技术达不到预期的效果	技术风险	30%	1	
缺少对工具的培训	技术风险	80%	3	
人员缺乏经验	项目风险	30%	2	
人员流动频繁	项目风险	60%	2	

说明:影响类别取值:1——灾难的;2——严重的;3——轻微的;4——可忽略的

2. 风险概率与风险分级

风险事件发生的概率或概率分布分析是进行工程项目风险估计的基础。一般而言，风险事件的发生概率或概率分布应由历史资料和数据来确定，即客观概率；当人们没有足够的历史资料和数据来确定风险事件的发生概率或概率分布时，可以利用理论概率分布或主观概率进行风险估计。

使用风险概率的数值对风险分级有着十分重要的意义。尽管行业或企业风险数据库基于大量项目的实例，可能对提供已知概率估计值有用，但对于个体来说，估算和应用概率是相当困难的。不过，大部分项目团队可以描述他们以往的经验，产生行业报告，并提供自然语言术语来映射数字概率范围。这可能和将（高，中，低）映射为不连续的概率数值（20%，50%，80%）一样简单，也可能和映射不同的自然语言术语一样复杂，例如"不一定高"、"不可能"、"很可能"、"几乎肯定"等，这些术语描述了概率中的不确定性。表 6-4 给出了一个 7 层概率分级的例子，由表中可以看出：当风险级别为 1 时表示风险发生可能性极高（几乎肯定发生），当风险级别为 7 时表示风险发生的可能性很低。

表 6-4 风险概率分级

概率范围	用来计算的概率值	自然语言表达	风险级别
1%～14%	7%	可能性很低	7
15%～28%	21%	可能性低	6
28%～42%	35%	可能性较低	5
43%～57%	50%	有一半可能	4
58%～72%	65%	可能	3
73%～86%	79%	非常可能	2
87%～99%	93%	几乎肯定	1

3. 风险参照系

风险事件发生的概率或概率分布是进行项目风险分析估计的基础。一种对风险评估的常用技术是定义风险的参照水准，对绝大多数软件项目来讲，风险因素——成本、性能、支持和进度就是典型的风险参照系。也就是说对成本超支、性能下降、支持困难、进度延迟都有一个导致项目终止的水平值。如果风险的组合所产生的问题超出了一个或多个参照水平值时，就终止该项目的工作。在风险分析中，风险水平参考值是由一系列的点构成的，每一个单独的点常称为参照点或临界点。如果某风险落在临界点上，可以利用性能分析、成本分析、质量分析等来判断该项目是否继续进行或终止项目。当超过它时会引起项目终止（阴影区域），如图 6-14 所示。实际上，参考水平很少能表示成如图所示的一条光滑曲线。在大多数情况下，它是一个区域，其中存在很多不确定性。

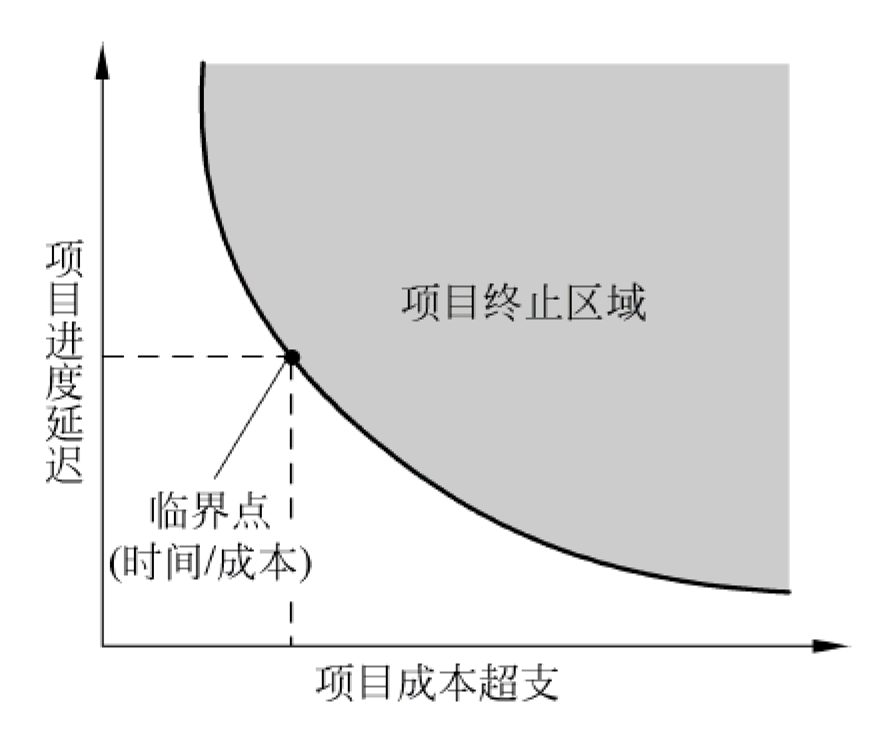


图 6-14 风险参照系

4. 专家估计法与德尔菲(Delphi)法

个体对风险因素或风险事件发生概率的判断可能主观性较大。为避免个体行为的偏差,使估计更符合客观实际,常需要充分利用专家们的集体智慧,由专家们来确定风险因素或风险事件的发生概率。

在进行成本估算时也用到德尔菲这种方法。德尔菲法采用匿名发表意见的方式,即专家(风险管理专家)之间不得互相讨论,不发生横向联系,只能与调查人员(风险管理人员)发生关系,通过多轮次调查专家对问卷所提问题的看法,经过反复征询、归纳、修改,直至专家意见趋于稳定,最后汇总成专家基本一致的看法,作为预测风险的结果。德尔菲技术有助于减少数据中的偏倚,并防止任何个人对结果不适当地产生过大的影响。

5. 决策树分析

决策树是对所考虑的决策以及采用这种或者那种现有方案可能产生的后果进行描述的一种图解方法(图 6-15)。它的预期成本是每个事件逻辑路径的概率和成本的乘积总和,其计算公式为:

$$\text{预期成本} = \sum(\text{路径概率})_i \times (\text{估算的路径成本})_i \tag{6-4}$$

其中:i 是决策树的某条路径。当所有路径的事件概率和成本全部量化之后,决策树的求解过程可得出每项方案的预期成本(或其他衡量指标)。

【例 6-1】 某软件企业准备开发一个新型软件产品,现有两种方案可供选择:第 1 种方案是在原有软件产品上进行功能更新;第 2 种方案是弃用旧产品,重新研制一种新软件产品。据分析测算,如果市场需求量大,更新旧的软件产品可获利 30 万元,生产新的软件产品可获利 50 万元。如果市场需求量小,更新旧的软件产品仍可获利 10 万元,生产新软件产品将亏损 5 万元(以上损益值均指一年的情况)。另据市场分析可知,市场需求量大的概率为 0.8,需求量小的概率为 0.2。试分析和确定哪一种生产方案可使软件企业年度获利最多?

解:① 绘制决策树,如图 6-15 所示。

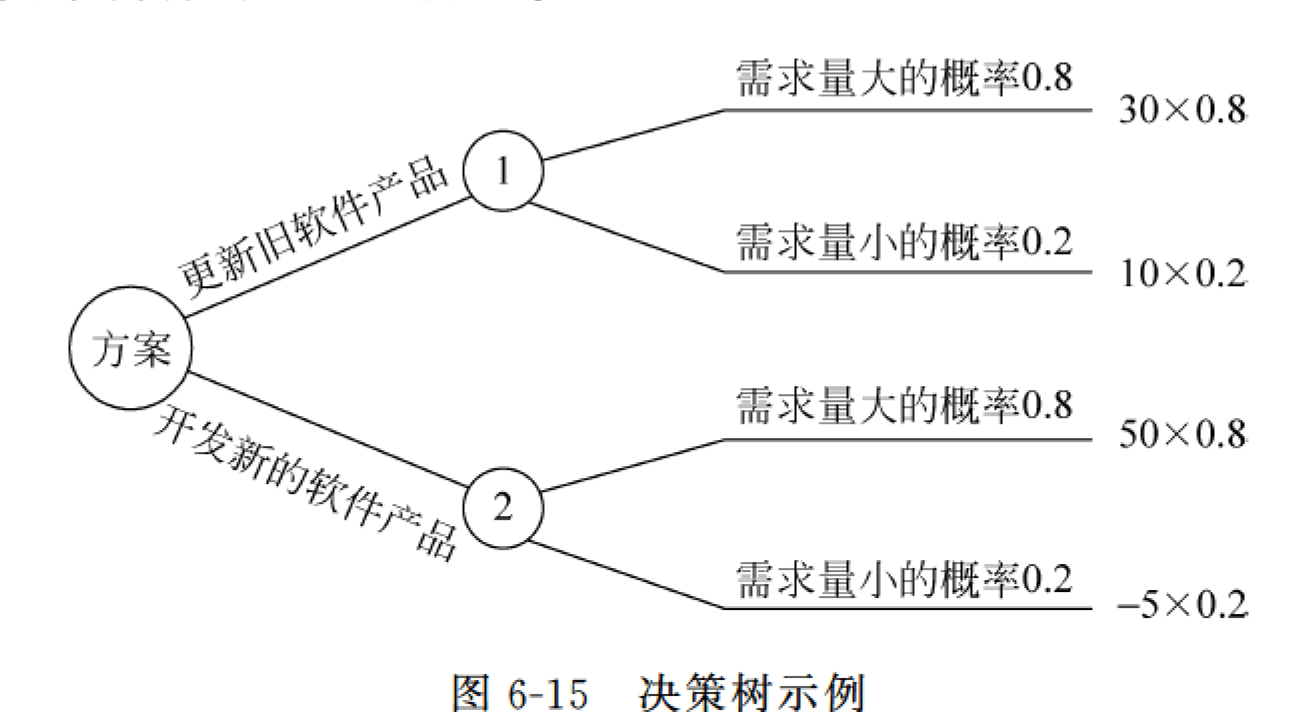


图 6-15　决策树示例

② 计算各结点的期望值

方案 1:30×0.8+10×0.2=26 (万元)

方案 2:50×0.8+(−5)×0.2=39(万元)

决策期望值为:max|26,39|=39(万元)

根据获利最多这一评价准则,合理的生产方案应为开发新的软件产品。

6. 标准离差率

标准离差也称为标准差(Standard Deviation),它反映的是各数据偏离平均数的距离(离均差)的平均数,标准差为方差(一般用 σ 表示)的平方根,用 S 表示。

标准离差率是标准离差与期望值之比。其计算公式为:

$$标准离差率 = 标准离差 / 期望值 \tag{6-5}$$

标准离差率是一个相对指标,是以相对数来衡量待决策方案的风险,尤其适用于期望值不同的决策方案风险程度的比较,可作为风险分析的工具之一。期望值不同的情况下,标准离差率越大,风险越大。反之,标准离差率越小,风险越小。如果资产的预期收益率相同则不需要计算标准离差率。

【例 6-2】 某企业拟进行一项存在一定风险的软件项目投资,有甲、乙两个方案可供选择:已知甲方案净现值的期望值为 1000 万元,标准离差为 300 万元;乙方案净现值的期望值为 1200 万元,标准离差为 330 万元。试比较两个方案的优劣。

解:当两个方案的期望值不同时,决策方案只能借助标准离差率这一相对数值。

由标准离差率计算公式可得:

甲方案标准离差率=300/1000=30%

乙方案标准离差率=330/1200=27.5%

显然甲方案的风险大于乙方案。

【例 6-3】 某企业有 A、B 两个投资项目,计划投资额均为 1000 万元,其收益(净现值)的概率分布如表 6-5 所示。

表 6-5 收益概率分布 金额单位:万元

项目状态	概率	A 项目净现值	B 项目净现值
乐观情况	0.2	200	300
一般情况	0.6	100	100
较差情况	0.2	50	−50

试完成以下计算:

(1) 分别计算 A、B 两个项目净现值的期望值。

(2) 分别计算 A、B 两个项目期望值的标准离差(标准差)。

(3) 判断 A、B 两个投资项目的优劣。

解:

(1) 计算两个项目净现值的期望值(风险期望)如下:

A 项目:$200\times0.2+100\times0.6+50\times0.2=110$(万元)

B 项目:$300\times0.2+100\times0.6+(-50)\times0.2=110$(万元)

(2) 计算两个项目期望值的标准离差(标准差):

A 项目:$S=\sqrt{(200-110)^2\times0.2+(100-110)^2\times0.6+(50-110)^2\times0.2}=48.99$

B 项目:$S=\sqrt{(300-110)^2\times0.2+(100-110)^2\times0.6+(-50-110)^2\times0.2}=111.36$

(3) 判断A、B两个投资项目的优劣

由于A、B两个项目投资额相同，期望收益也相同，而A项目风险相对较小(其标准离差小于B项目)，故A项目优于B项目。

6.5　风险规划

风险规划是风险管理过程中的第3个阶段。由项目团队执行的计划工作将分类风险清单转化为行动计划。规划包含为最大风险展开的详细策略和行动、风险行为分级以及综合风险管理计划的创建。

6.5.1　风险规划过程

风险规划过程的活动是将按优先级排列的风险列表转变为风险应对计划所需的任务。风险规划过程的IDEF0图如图6-16所示。

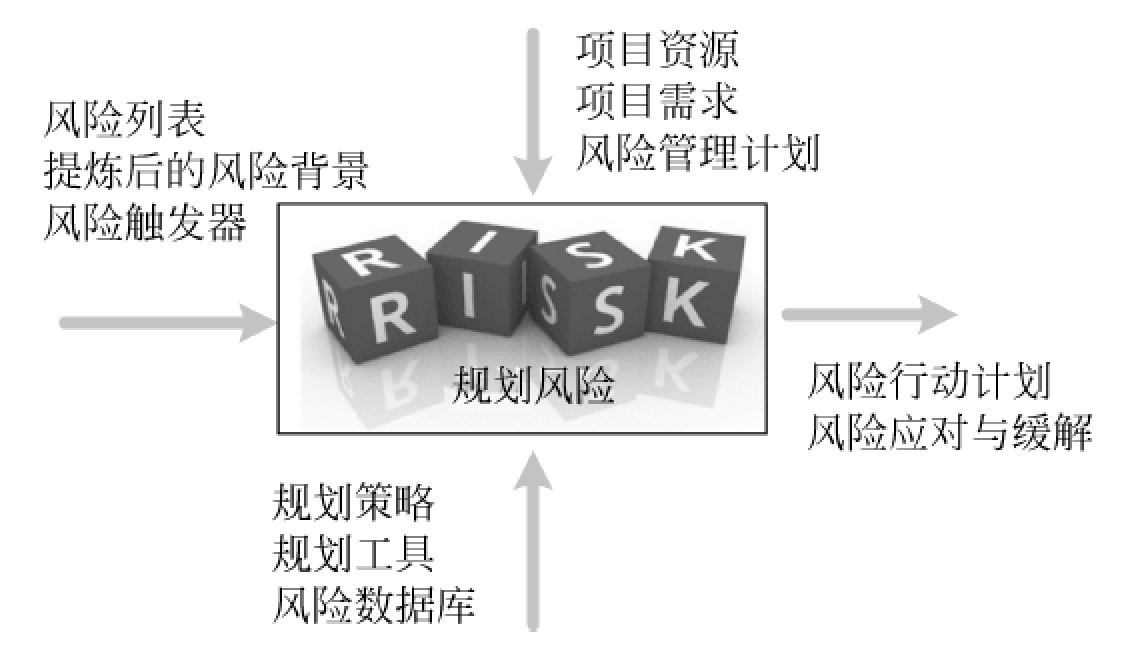


图6-16　风险规划过程

1. 过程控制

项目资源、项目需求和风险管理计划调节风险规划过程。风险管理计划的重要内容包括角色和职责，风险分析定义，低风险、中等风险和高风险的风险限界值，进行项目风险管理所需的费用和时间。风险管理计划的某些要素是风险应对规划的依据，这些要素包括低、中、高风险的风险限度，这些风险限度能够帮助我们很好地了解那些需要采取应对措施的风险，以及风险应对规划中的人员分配、进度安排和预算制定。

2. 过程输入

(1) 风险列表：包含所有已知风险按优先次序排列的列表。

(2) 提炼后的风险背景：围绕在风险声明周围的间接信息(事件、情况、约束、假设、环境、有影响的因素和相关的问题)，可从风险数据库获得这些信息。

(3) 触发器：启动、解除或延缓风险行动计划的装置。当风险值达到风险阈值时就可能启动触发器，触发器的状态可由项目风险监控制度设定。

3. 过程输出

过程输出包括以下内容。

(1) 风险应对与缓解：风险应对策略是指用来应对风险的一套选择和方案，包括风险规避、风险转移、风险接受、风险减缓。每个策略应包括目标、约束和备用方案。

(2) 风险行动计划：风险行动计划详细说明了所选择的风险应对途径。它将途径、所需的资源和批准权编写为文档。

4. 过程机制

机制可以是方法、技巧、工具或其他为过程活动提供结构的手段。风险应对策略和风险数据库都是风险应对过程的机制。风险数据库包含风险行动计划、相关的场景和阈值。

6.5.2 风险规划的工具与技术

通常，使用3种策略应对可能对项目目标存在消极影响的风险或威胁。这些策略分别是风险规避、风险转移、风险接受、风险减缓。

(1) 风险规避。风险规避是在项目早期的计划阶段经常使用的一种有效手段。风险有可能通过下列行动被消除：缩小项目目标或功能的范围，延长进度等。或者采用一种进化的开发方法以利用其临时解决该风险问题所具备的初步能力。

(2) 风险转移。与风险规避一样，风险转移也是项目开始阶段所用的一种有效途径。例如，把一个具有高风险的功能转移到一个能够成功实现它的相关项目或系统中。

(3) 风险接受。有时，由于政策、市场或客户的需求，必须接受风险的事实以处理风险。对该风险以及它对项目的影响必须进行持续地监视和报告。这种情况下，项目管理者要接受所涉及的风险，并且承认结果发生的可能性。这时，风险事实会给项目计划或预算带来影响，所以需要估计出这一部分的成本和工作量，把它作为管理储备估计的一部分，以留有足够的资金，在风险发生时可以使用这一部分储备。

(4) 风险减缓。风险减缓是指建立一种行动计划以阻止风险的发生或者是当风险发生时减少它对项目的影响。

制定风险缓解措施时，要参考已排序的风险清单表。优先级越高的风险，优先保证缓解措施所需资源。对于优先级高的风险，可以考虑制定多个缓解措施。

6.6 风险监控

风险监控就是通过对风险识别、分析、规划和应对的全过程进行管理和监控，从而保证风险管理能达到预期的目标，它是项目实施过程中的一项重要工作。

6.6.1 风险监控过程

风险监控过程活动包括监视项目风险的状况，保持项目管理在预定的轨道上进行。其内容包括监测风险是否已经发生，仍然存在还是已经消失；检查风险应对策略是否有效，监

控机制是否在正常运行,并不断识别新的风险,及时发出风险预警信号并制定必要的对策措施。

风险监控过程的 IDEF0 图如图 6-17 所示。



图 6-17 风险监控过程

1. 过程控制

和控制风险规划过程一样,项目资源、项目需求、风险管理计划同样约束着风险监控过程。

2. 过程输入

风险背景、风险行动计划、风险应对计划作为过程输入。

3. 过程输出

风险监控标准、应变措施、控制行动、变更请求等是风险监控过程的输出,主要内容如下。

(1) 风险监控标准:主要指项目风险的类别、发生的可能性和后果。

(2) 应变措施:就是消除风险事件时所采取的未事先计划到的应对措施。这些措施应有效地记录,并融入项目的风险应对计划中。

(3) 控制行动。控制行动就是实施已计划了的风险应对措施。

(4) 变更请求。对风险做出反应会导致风险应对计划的改变,这种变更应以规范的方式提出。

(5) 修改风险应对计划。当预期的风险发生或未发生时,当风险控制的实施消减或未消减风险的影响或概率时,必须重新对风险进行评估,对风险事件的概率和价值以及风险管理计划的其他方面做出修改,以保证重要风险得到恰当控制。

4. 过程机制

机制可以是方法、技巧、工具或为过程活动提供的其他手段。风险监控方法、风险监控工具和风险数据库都是风险监控过程的机制。风险监控工具的使用使监控过程自动化、高效化。

6.6.2 风险监控的技术与方法

风险管理是一个连续的过程,因此在软件项目的实施过程中需要遵循预先制定的计划定期监督风险。风险监控还没有一套公认的、单独的技术可供使用,其基本目的是以某种方式监控风险,保证项目可靠、高效地完成项目目标。风险应从以下 3 个方面进行监控。

(1) 监控风险的状态。监控风险的状态并对风险缓解措施的执行情况进行跟踪,将风险状态和缓解措施执行情况记录于文档之中。

风险状态包括:①风险被缓解;②风险已发生,转入应对;③监控中,缓解措施正在实施;④监控中;⑤新识别风险。

(2) 应急计划的制定与执行。应急计划是为控制项目实施过程中有可能出现或发生的特定情况做好准备。由于软件项目风险的复杂性,必须对项目实施过程中各种风险(已识别的或潜在的)进行系统管理,并对项目风险可能的各种意外情况进行有效管理。因此,制定应对各种风险的应急计划是项目风险监控的一个重要工作,也是实施项目风险监控的一个重要途径。

触发器(trigger)在项目风险监控中是一个十分有用的概念,一个有效的应急计划往往把风险看做是由某种"触发器"引起的,即项目中的风险存在着某种因果关系。触发器可提供 3 种基本的控制功能:第一是激活,触发器提供再次访问风险行动计划(或对照计划取得的进展)的警铃;第二是解除,触发器可用于发送信号,终止风险应对活动;第三是挂起,触发器可用于暂停执行风险行动计划。

(3) 风险持续管理。面向持续改进的风险管理必须因项目管理阶段不同、主体间差异而有所不同,贯穿于项目管理活动的始终(图 6-18)。

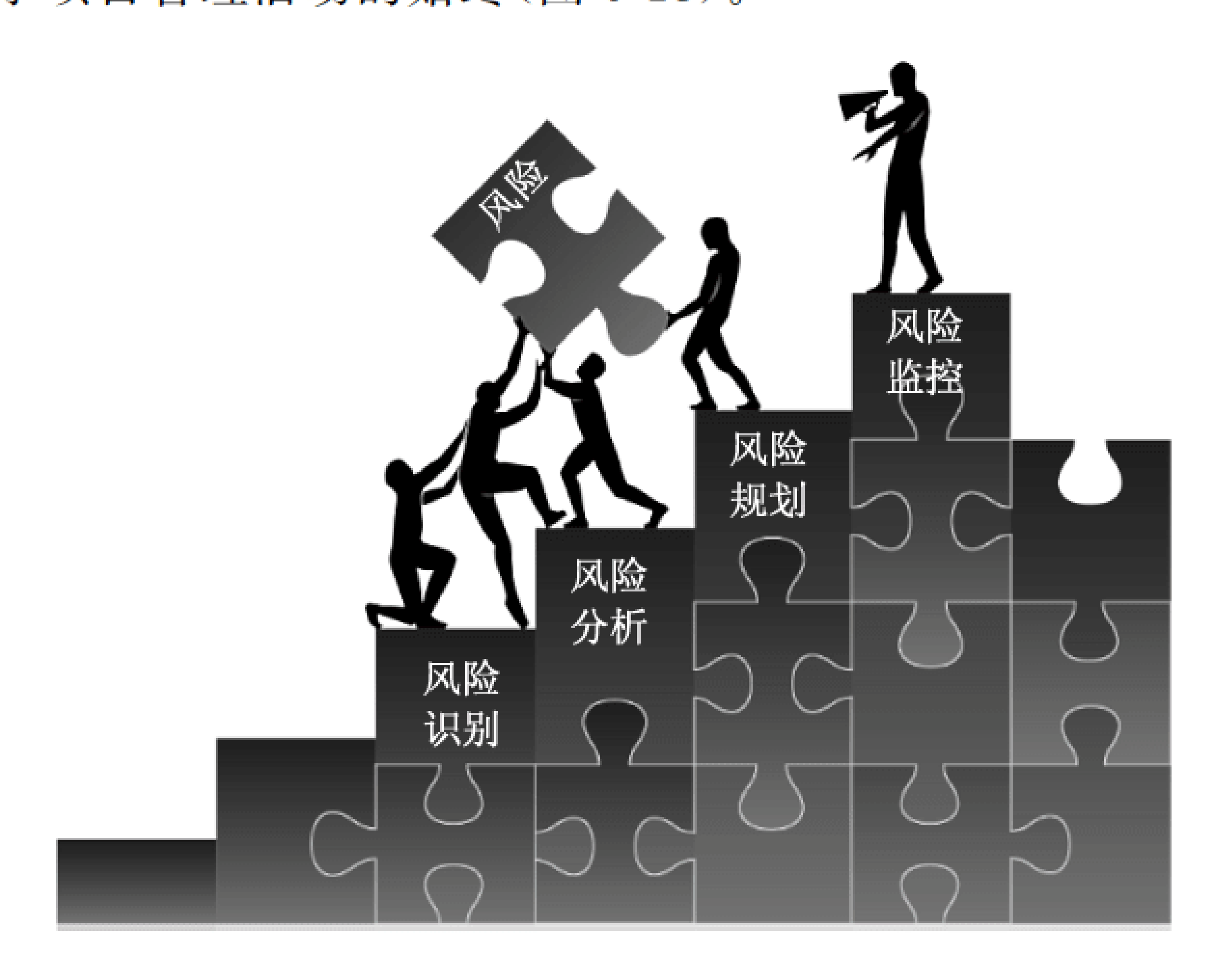


图 6-18 软件风险的持续管理

通常,项目风险管理持续改进的主要功能如下。

① 识别。在风险成为问题前,搜寻并定位风险。

② 分析。通过系统化分类、估计和理解以降低不确定性并提供计划和行动框架的流程,把风险转换成决策信息。

③ 计划。把风险信息转化为决策和适当的消减行动(现在与未来)以及实施。

④ 跟踪。通过项目生命周期来监控风险指标和风险消减计划。

⑤ 控制。纠正与计划执行的偏差。

⑥ 沟通。贯穿所有以上提供信息和反馈的功能给所有项目相关利益人。

6.6.3 风险监控与管理计划——RMMM 计划

风险管理策略可以包含在软件项目计划中,或者风险管理步骤也可以组织成一个独立的风险缓解、监控和管理计划(RMMM 计划(Risk Mitigation,Monitoringand Management Plan)。RMMM 计划将所有风险分析文档化,并由项目管理者作为整个项目计划中的一部分来使用。一旦建立了 RMMM 计划,且项目开始启动,则风险缓解监控步骤也开始了。

RMMM 计划的大纲如下。

Ⅰ. 引言

文档的范围和目的

主要风险综述

责任

a. 管理者

b. 技术人员

Ⅱ. 项目风险表

终止线之上所有风险的描述

影响概率及影响的因素

Ⅲ. 风险缓解、监控和管理

缓解

一般策略

缓解风险的特定步骤

监控

被监控的因素

监控办法

管理

意外事件计划

特殊的考虑

Ⅳ. RMMM 计划的迭代时间安排表

Ⅴ. 总结

本 章 小 结

风险是阻碍事物运动发展的客观存在,是事物发生与否的某种不确定性。从认知学的角度讲,风险的损害发生与否,损害的程度取决于人类主观认识和客观存在之间的差异性。在这个意义上说,风险指在一定条件下特定时期内,预期结果和实际结果之间的差异程度。

软件项目风险管理是软件项目管理的重要内容。风险管理是一个管理过程,包括对风险的定义、测量、评估和应对风险的策略。目的是将可避免的风险、成本及损失极小化。理想的风险管理,事先已排定优先次序,可以优先处理引发最大损失及发生几率最高的事件,其次处理风险相对较低的事件。

对风险管理研究的方法采用定性分析方法和定量分析方法。定性分析方法是通过对风险进行调查研究,做出逻辑判断的过程。定量分析方法一般采用系统论方法,将若干相互作用、相互依赖的风险因素组成一个系统,抽象成理论模型,运用概率论和数理统计等数学工具定量计算出最优的风险管理方案的方法。

软件项目风险是指在软件开发过程中遇到的预算和进度等方面的问题以及这些问题对软件项目的影响。软件项目风险会影响项目计划的实现,如果项目风险变成现实,就有可能影响项目的进度,增加项目的成本,甚至使软件项目不能实现。如果对项目进行风险管理,就可以最大限度地减少风险的发生。成功的项目管理一般都对项目风险进行了良好的管理。因此任何一个系统开发项目都应将风险管理作为软件项目管理的重要内容。

在项目风险管理中,存在多种风险管理方法与工具,软件项目管理只有找出最适合自己的方法与工具并应用到风险管理中,才能尽量减少软件项目风险,促进项目的成功。

在本章中我们定义了风险管理的概念和过程,管理过程包括风险识别、风险分析、风险规划和风险监控。每一个步骤都定义了 IDEF0 过程,用活动是什么以及如果通过机制(方法、技术和工具)将输入转变为输出来表示它。

风险存在于软件项目开发的整个过程,风险管理必须根据风险变化及时动态调整,达到风险预测、实时响应、降低风险的良性循环能力,保证项目开发成功。风险管理活动会增加许多项目管理的工作量,但这是值得的。引用中国 2500 多年前的兵法家孙子的话:"知己知彼,百战不殆。"对于软件项目管理者而言,这个"彼"指的就是风险。

思考与练习

1. 什么是风险?项目风险有哪些特征?

2. 什么是软件项目风险管理?试举例说明许多信息系统失败于不恰当的项目风险管理。

3. 风险揭露用来量化风险,为所有已知风险提供了相对的参考,试叙述风险揭露的形式化表示。

4. 软件风险的类型有几种?试分析各类风险的特征。

5. 软件风险管理的体系框架有哪些?试分析各种体系的特点与相同点。

6. 软件风险管理的一般过程包括哪些阶段?

7. 什么是 IDEF0 图?该图如何通过控制、输入、输出和机制来描述风险管理过程?

8. 试根据风险识别的 IDEF0 图,说明风险识别的基本过程。

9. 风险识别的方法与工具有哪些?试用因果关系分析法画出导致项目进度延迟的主要因素。

10. 试根据风险分析的 IDEF0 图,说明风险分析的基本过程。

11. 风险分析的技术与工具包括哪些?风险列表的作用是什么?

12. 风险事件发生的概率或概率分布分析是进行工程项目风险估计的基础,试解释风

险参照系的作用。

13. 什么是决策树分析？其计算公式是如何表示的？

14. 什么是标准离差率，它对于方案风险决策有何意义？

15. 对于多个投资方案而言，无论各方案的期望值是否相同，标准离差率最大的方案是否一定是风险最大的方案。

16. 试根据风险规划的 IDEF0 图，说明风险规划的基本过程。

17. 试根据风险监控的 IDEF0 图，说明风险监控的基本过程。

18. RMMM 计划由几部分内容组成？

19. 某软件小组计划项目中采用 50 个可复用的构件，每个构件平均是 100 LOC，本地每个 LOC 的成本是 13 元人民币。预定要复用的软件构件中只有 50%将被集成到应用中，剩余功能必须定制开发，风险概率为 60%。试求该项目风险的风险曝光度。

20. 某公司的历史数据表明：每 KLOC 的错误率为 0.0036，每个错误会使公司平均损失 10 000 元。一种新的评审技术表明，每评审 100KLOC 的程序需要花费 1000 元，并减少错误率 50%。当前项目的大小估算为 50KLOC。试计算每种方法的风险预期，新的评审方法值得采用吗？

21. 在一个项目中，计划有 60 个可复用的软件构件，平均每个构件的程序行数是 100 LOC。本地数据表明，每 LOC 的成本是 50 元。现已知存在一种项目风险，即计划复用的软件构件中可能只有 60%将集成到应用系统中，这种风险发生的概率是 80%。试计算风险预期值。

22. 已知甲方案投资收益率的期望值为 15%，乙方案投资收益率的期望值为 12%，两个方案都存在投资风险。在比较甲、乙两方案风险大小时应采用什么分析指标？

23. 某软件组织准备开发一款新的网络浏览器，试用风险检查表与风险列表技术对新浏览器的可能风险进行识别与分析。

(1) 在风险检查表中按风险类别列出一些可能危及项目成功的风险条目。

风险类型	序号	风险检查条目
项目风险	1	
	2	
商业风险	1	
	2	
技术风险	1	
	2	

(2) 识别每个类别中的具体风险，并标识风险类别、发生的概率、产生的影响和风险缓解计划(RMMM)。

风险条目	类别	概率	影响	RMMM

24. 某企业有甲、乙两个投资项目，计划投资额均为1000万元，其投资回报率的概率分布如下表所示：

项目状态	概率	甲项目	乙项目
乐观情况	0.3	20%	30%
一般情况	0.5	10%	10%
较差情况	0.2	50%	-5%

要求：

(1) 分别计算甲乙两个项目投资回报率的期望值。

(2) 分别计算甲乙两个项目投资回报率的标准差。

(3) 比较甲乙两个投资项目风险的大小。

第2篇

结构化开发方法



本篇内容：

- 面向过程的结构化分析
- 面向过程的结构化设计
- 面向过程的结构化实现
- 软件的技术度量

第7章 面向过程的结构化分析

分析显而易见的事情需要非凡的思想。

——Alfred North Whitehead[①]

7.1 系统工程

7.1.1 系统论

系统思想源远流长，古今中外都有，在《申鉴·时事》中有一段话是这样的："有鸟将来，张罗待之，得鸟者一目也。今为一目之罗，无时得鸟矣。"这个故事中的捉鸟人，只看到一只网眼捉到了鸟，而没有认识到整张网的作用，于是抛弃了整张网，显然也就再捉不到鸟了。

"系统"一词，来源于古希腊语，是部分构成整体的意思。根据对系统研究角度的不同，人们给出了许多"系统"的定义，如"系统是有组织的和被组织化的全体"，"系统是有联系的物质和过程的集合"，"系统是许多要素保持有机的秩序，向同一目的行动的东西"等。人们一般认为是美籍奥地利人、理论生物学家 L. Von. Bertalanffy 创立了系统论学科。他在1952 年发表"抗体系统论"，提出了系统论的思想。1973 年他又提出了一般系统论原理，奠定了这门科学的理论基础。他的一般系统论则试图给出一个能描述各种系统共同特征的一般的系统定义，通常把系统定义为：由若干要素以一定结构形式联结构成的具有某种功能的有机整体。在这个定义中包括了系统、要素、结构、功能 4 个概念，表明了要素与要素、要素与系统、系统与环境 3 方面的关系。世界上任何事物都可以看成是一个系统，系统是普遍存在的，大至渺茫的宇宙，小至微观的原子，一粒种子、一群蜜蜂、一台机器、一个工厂、一个学会团体都是系统，整个世界就是系统的集合。

系统论认为，所有系统共同的基本特征包括整体性、关联性、等级结构性、动态平衡性、时序性等。这些既是系统所具有的基本思想观点，也是系统方法的基本原则，表现了系统论不仅是反映客观规律的科学理论，也具有科学方法论的含义，这正是系统论这门科学的特点。它的核心思想是系统的整体观念。它强调，任何系统都是一个有机的整体，它不是各个部分的机械组合或简单相加，系统的整体功能是各要素在孤立状态下所没有的性质。系统

---

① Alfred North Whitehead(1861—1947)，英国数学家、教育家和著名形而上学家。他生于英国肯特郡，是著名数学家罗素的朋友，他与罗素合著的《数学原理》标志着人类逻辑思维的空前进步，被称为永久性的伟大学术著作之一。他创立了 20 世纪最庞大的形而上学体系。

论反对那种认为要素性能好，整体性能一定好，以局部说明整体的机械论的观点。同时认为，系统中各要素不是孤立地存在着，每个要素在系统中都处于一定的位置上，起着特定的作用。要素之间相互关联，构成了一个不可分割的整体。要素是整体中的要素，如果将要素从系统整体中割离出来，它将失去要素的作用。例如，在一个 ATM 自动柜员机系统中，数字键盘是一个用于输入信息的设备，它和 ATM 自动柜员机系统的主机相连，如果将数字键盘从主机上拿下来，单独的数字键盘就不能发挥其输入信息的功能，反过来说，被拿掉数字键盘后的 ATM 自动柜员机（不考虑其触摸屏功能）也无法正常工作。

系统论也可以说是一种方法论。系统论的基本思想方法就是：以所研究和处理的对象作为一个系统，分析系统的结构和功能，研究系统、要素、环境 3 者的相互关系和变动的规律性，并优化系统观点看问题。系统论的任务，除了要认识系统的特点和规律，更为重要的是，要利用这些特点和规律去控制、管理、改造或创造某个系统，使它的存在与发展合乎人的目的需要。也就是说，研究系统的目的在于调整系统结构和各个要素之间的关系，以优化该系统。

7.1.2 系统工程

系统工程（Systems Engineering）是系统科学的一个分支，实际是系统科学的实际应用，可以用于一切有大系统的方面，包括人类社会、生态环境、自然现象、组织管理等，如环境污染、人口增长、交通事故、军备竞赛、化工过程、信息网络等。系统工程是以大型复杂系统为研究对象，按一定目的进行设计、开发、管理与控制，以期达到总体效果最优的理论与方法。系统工程还可以用于化工生产设计过程优化控制、信息网络运筹等多个方面。

系统工程的目的是解决总体优化问题，从复杂问题的总体入手，认为总体大于各部分之和，各部分虽较劣但总体可以优化。有些问题，如电话网络，不能只研究个别电话的质量问题，必须从总体网络入手。

因为系统工程要研究各类系统的特性和共性，因此所使用的技术内容很多。系统工程是一门工程技术，但是，系统工程又是包括了许多类工程技术的一大工程技术门类，涉及范围很广，不仅要用到数、理、化、生物等自然科学，还要用到社会学、心理学、经济学、医学等与人的思想、行为、能力等有关的学科。系统工程涉及的主要技术内容有运筹学、概率论和数理统计学、现代化科学管理技术。

（1）运筹学所要解决的问题，是在既定条件下对系统进行全面规划，统筹兼顾，以期达到最优的目标。运筹学既是一门理论科学，又是一门应用科学。运筹学是系统工程的基础，系统工程则是这门科学理论的具体运用。运筹学的主要分支有规划论、对策论、排队论、决策论、库存论、可靠性理论、网络计划法等。

（2）概率论是研究随机性或不确定性等现象的数学，广泛应用于概率型模型的描述。数理统计学通过对某些现象的频率的观察来发现该现象的内在规律性，并做出一定精确程度的判断和预测，用来研究取得数据、分析数据和整理数据的方法。

（3）现代化科学管理技术。系统工程所面临的基本问题是：怎样把比较笼统的初始规划逐步地变成为很多具体规划工作，以及怎样把这些工作最终综合成一个技术上合理、经济上合算、研制周期短、能协调运转的实际系统，这样复杂的总体协调工作就需要运用现代化的科学管理技术。现代化的科学管理技术既是技术问题，更是一个经济问题，它是符合科学

规律、运用先进的科学技术和经济思想，把整个生产管理组织起来的一门科学。

7.1.3 系统工程层次结构

有很多软件开发人员只见树木，不见森林，最终导致项目失败。开发软件系统，需要有全局的眼光，检查整个业务和产品领域，保证能在“全局视图”上理解产品所在的位置。然后重点分析所关心的具体领域，在这个确定的领域上分析出所需系统的要素(如数据、软件、硬件、人员等)，然后对所需系统进行需求分析、系统设计和构建。在系统全局视图上包含业务和产品域，而具体的技术活动在详细视图层面上实现①。

系统工程层次结构如图 7-1 所示。

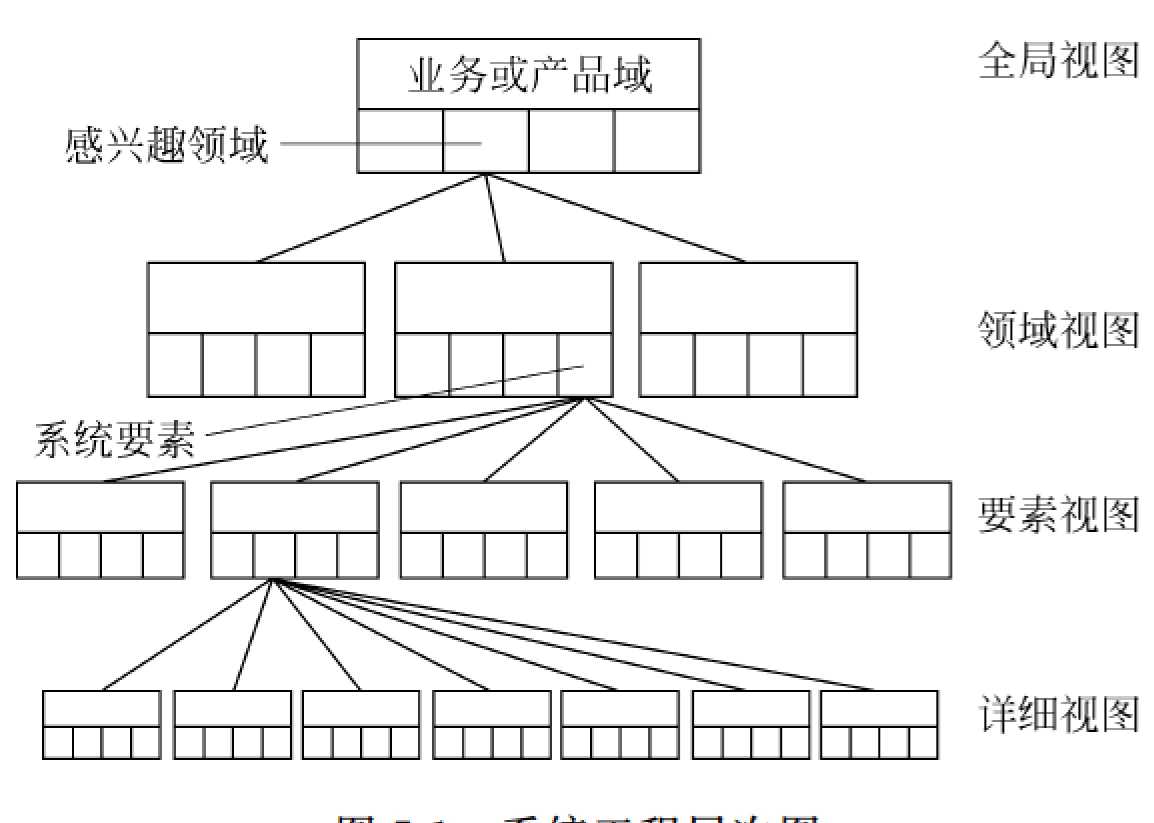


图 7-1 系统工程层次图

从每个视图的组成来看，领域集合 D_i 组成全局视图(World View，WV)，即：

$$WV=\{D_1, D_2, \cdots, D_n\}$$

而每个领域都由特定要素(E_j)组成，各自在完成某领域或其组成部分目标的过程中扮演一些角色。

$$D_i=\{E_1, E_2, \cdots, E_M\}$$

最后，每种要素通过完成特定功能的构件(C_k)来实现

$$E_i=\{C_1, C_2, \cdots, C_k\}$$

每个构件可以是类、对象、模块、一段计算机程序，甚至是一条编程语句。

系统工程层次结构给出了概括性的框架，而需要对该框架进行具体的实例化，在实际应用中有多种框架可以考虑，以利于理解正确的客户需求。这里介绍两种常用框架：业务过程工程、产品工程②。当工作集中在某业务企业时，业务过程工程就会发挥作用；而关注产品生产的过程称为产品工程。通俗地讲，就是我们是做项目还是做产品，做项目选择业务过程工程，做产品选择产品工程。

---

① Roger S. Pressman. 软件工程——实践者的研究方法. 郑人杰，马素霞，白晓颖，等译. 北京：机械工业出版社，2007

② Roger S. Pressman. 软件工程——实践者的研究方法. 郑人杰，马素霞，白晓颖，等译. 北京：机械工业出版社，2007

7.1.4 业务过程工程

业务过程工程是要从公司信息技术需求的全局角度出发，定义出一个能有效利用信息的体系。业务过程工程为建立实施计算架构的总体计划提供了一种方法。业务过程工程可以让软件工程师更加深刻地理解所开发系统在企业信息建设中所处的位置，更有利于企业有效利用信息的软硬件资源，业务过程工程层次图如图 7-2 所示。

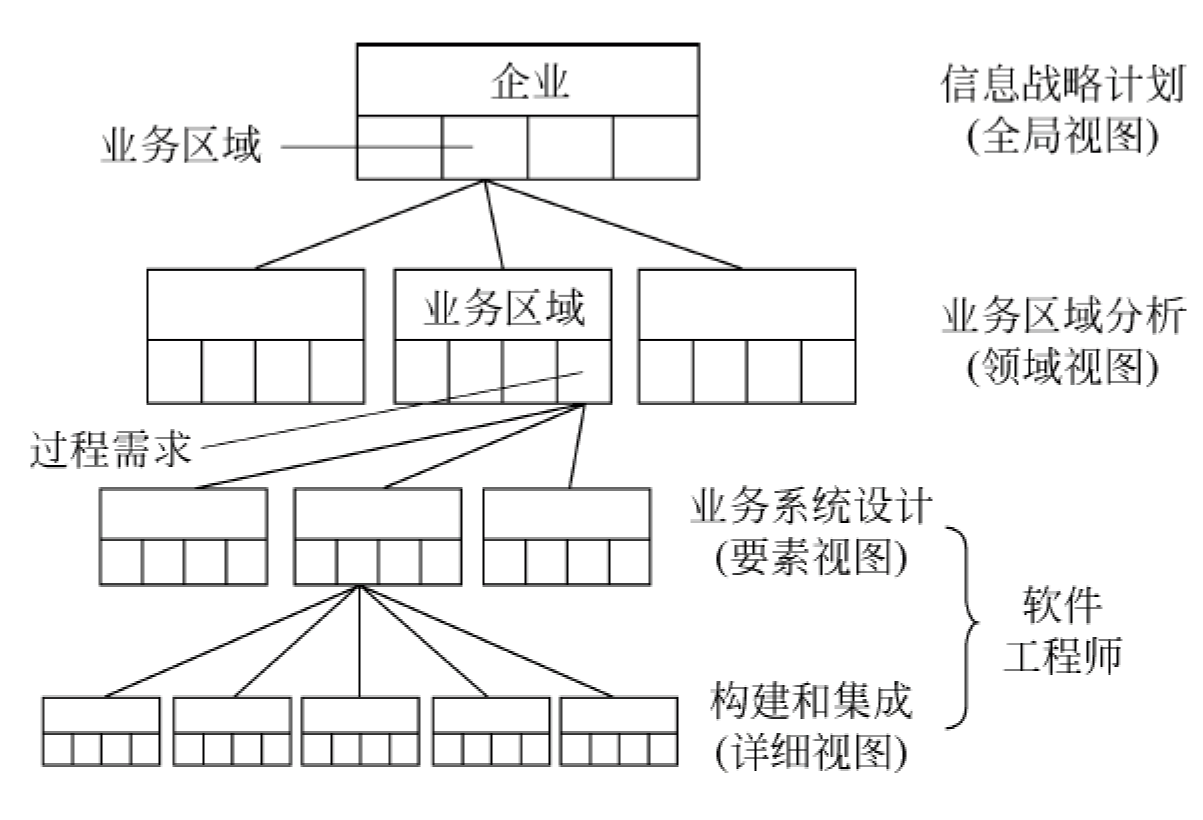


图 7-2 业务过程工程层次图

在这个体系中需要分析和设计 3 种不同的架构：数据架构、应用架构、技术基础设施。

数据架构为业务或业务功能的信息需求提供了框架，单独建立的框架模块是被业务所用到的数据对象。一个数据对象包括用于定义不同侧面的属性集、质量、特征或数据描述符，可以理解为 E-R 模型中的实体。数据对象集确定了，就需要确定数据对象之间的关系，关系表明对象之间是如何联系的。例如，在一个图书借阅管理系统中，存在图书和用户对象，他们通过借阅和归还联系起来，即图书被用户借阅或归还。在业务活动过程中，以数据库的方式将对象组织起来，以提供业务活动所需的信息。

应用架构是数据架构范围内执行数据或应用处理的程序系统，以达到企业的某些业务目标或目的。例如，对于一个图书馆，其原来的图书管理业务模式是通过卡片或纸片来索引和管理图书数据，管理者希望在引入信息系统后，通过录入、修改、删除书籍程序，实现对图书数据的电子化管理，通过借阅和归还程序来实现图书借阅电子化管理，通过录入、修改、删除用户程序，来实现对借阅者的电子化管理。应用架构将角色和尚未实现自动化的业务规程联系在一起。

有数据架构和应用架构还是不够的，就好比要从某地运货到另一个地方，货物相当于数据架构的具体内容，而货车相当于应用架构的具体内容，而光有这两点是不够的，还需要一些基础设施，如高速公路、加油站等。同样的道理，业务过程工程也需要技术基础设施，技术基础设施为数据架构和应用架构提供基础，基础设施包括用来支持应用和数据的硬件和软件，包括计算机、操作系统、网络、通信链路、存储技术和用于实现这些技术的架构(如客户/服务器)。

7.1.5 产品工程

产品工程的目的是将用户期望的已定义的一组能力转变成真实产品。产品工程和业务过程工程一样，也需要定义出架构和基础设施。软件、硬件、数据(数据库)以及人员组成了

4个不同的系统构件。而文档、CD-ROM、视频这些能够将各种构件联系在一起的技术和信息，就是基础设施。

如图7-3所示产品工程层次图，由客户引出的全局需求确定全局视图。全局需求包括信息和控制要求、产品功能和行为、产品整体性能、设计和接口约束条件以及其他特殊要求。这些需求明确以后，需求分析的工作就是将这些全局需求分配到软件工程、硬件工程、数据(数据库)工程以及人员工程这4个不同的系统构件中。

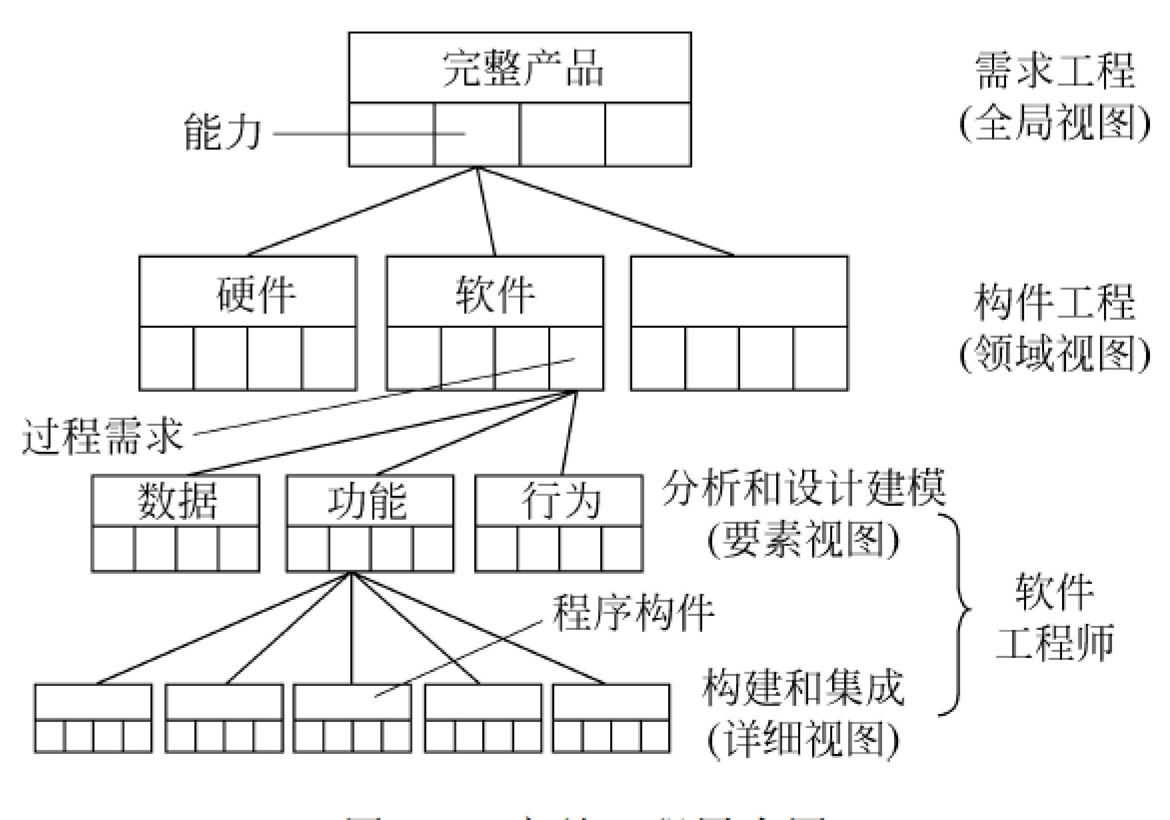


图7-3 产品工程层次图

分配工作的开始，意味着系统构件工程的开始。为了提高效率，系统构件工程往往采用并发的工作方式，同步地分别处理软件工程、硬件工程、人员工程和数据库工程这些系统构件。在这个过程中，建立工程规范和维持构件相互间的积极沟通是至关重要的，需求分析的部分作用是建立便于沟通的接口机制。同时为了更准确地分配和描述，每个工程规范都用特定领域的观点来看待。

应用于已分配构件的工程规范是产品工程的要素视图。具体来说，软件工程包括分析、设计建模、编码、测试和支持任务等活动。其中，分析任务模型是将需求分配到数据、功能和行为表示等要素中，设计建模是将分析模型映射到数据设计、结构设计、接口设计和软件构件级设计中。

7.2 需求分析

为什么要进行需求分析？

如果有这样一种情况，我们付出了大量的人力和费用将软件开发出来，结果这个软件产品不能满足用户的需要，没有人愿意使用，那么之前所投入的人力和费用都将付之东流，从而要重新开发。例如，用户需要一个网络版的进销存软件，而你匆匆忙忙地开始软件开发，没有向用户确认软件使用的范围，自以为是地认为所开发的软件是在一台计算机上使用的，而非是多人使用，当你花了大量精力终于开发完成，兴冲冲地向用户提交软件时才发现出了问题，那时候你真是后悔莫及。

软件的最终目的是用来解决用户的某些问题，这需要软件技术人员将用户的这些问题进行转化，以软件的形式来加以解决。通常情况是软件人员精通计算机技术，却对用户的业

务领域不了解或了解得不深入；同时，用户可能清楚自己的业务（当然有时也不一定了解，就有这么一个例子，某公司在做一个大型税务征管系统时，曾经向一个征收科科员了解需求，其中涉及报表部分的需求，这个科员很热情也很确定地告诉分析人员这部分的需求，分析人员一再确认这部分需求是否就是这样了，他回答说：是的。结果，分析人员将系统开发出来后，拿给这个科员的领导一看，结论是报表部分需要修改，为什么会这样呢？难道这个科员给出的需求是错误的？当然不是，原因在于科员的视角和管理者的视角是有差异的，不同层面上的用户，给出的需求也可能会有冲突），却又不太了解计算机技术。面对同一个问题，软件人员与使用者之间可能有认识上的不同。通常在软件设计之前，我们需要对软件问题进行准确、全面、系统的需求分析。

需求分析就是要彻底理解要解决的问题，真正明确用户需求。在解决问题之前要理解问题，只有真正地理解问题才能更好地解决问题。需求分析就是软件人员和用户之间进行交流，以理解问题。需求分析是软件开发工程中的一个重要阶段，在这个阶段中，软件系统分析人员研究用户在需要解决问题上的意愿，分析出软件系统应该达到的目标，以及有哪些功能要求、性能要求等。

因为需求分析具有决策性，方向性，策略性的作用，所以它在软件开发的过程中起到至关重要的作用。我们一定要对需求分析引起足够的重视，尤其在大型软件系统的开发中，更是如此。

7.2.1 需求分析概述和特点

1. 需求分析概述

在软件工程中，需求分析指的是在建立一个新的或改变一个现存的计算机系统时描写新系统的目的、范围、定义和功能时所要做的所有的工作。需求分析是软件工程中的一个关键过程。在这个过程中，系统分析员和软件工程师确定顾客的需要。只有在确定了这些需要后他们才能够分析和寻求新系统的解决方法。

在软件工程的历史中，很长时间里人们一直认为需求分析是整个软件工程中最简单的一个步骤，但在过去10年中越来越多的人认识到它是整个过程中非常关键的一个过程。假如在需求分析时分析者们未能正确地认识到顾客的需要，那么最后的软件实际上不可能达到顾客的需要，或者软件无法在规定的时间里完工。

2. 需求分析的特点

需求分析是一项重要的工作，也是最困难的工作。该阶段工作有以下特点。

(1) 用户与开发人员很难进行交流

在软件生存周期中，其他4个阶段都是面向软件技术问题，只有本阶段是面向用户的。需求分析是对用户的业务活动进行分析，明确在用户的业务环境中软件系统应该“做什么”。但在开始时，开发人员和用户双方都不能准确地提出系统要“做什么”。因为软件开发人员不是用户问题领域的专家，不熟悉用户的业务活动和业务环境，又不可能在短期内搞清楚；而用户不熟悉计算机应用的有关问题。由于双方互相不了解对方的工作，又缺乏共同语言，所以在交流时存在着隔阂。

（2）用户的需求是动态变化的

对于一个大型而复杂的软件系统，用户很难精确完整地提出它的功能和性能要求。一开始只能提出一个大概、模糊的功能，只有经过长时间的反复认识才逐步明确。有时进入到设计、编程阶段才能明确，更有甚者，到开发后期还在提新的要求。这无疑给软件开发带来困难。

（3）系统变更的代价呈非线性增长

需求分析是软件开发的基础。假定在该阶段发现一个错误，解决它需要用 1 小时的时间，到设计、编程、测试和维护阶段解决，则要花多很多的时间。

因此，对于大型复杂系统而言，首先要进行可行性研究。开发人员对用户的要求及现实环境进行调查、了解，从技术、经济和社会因素 3 个方面进行研究并论证该软件项目的可行性，根据可行性研究的结果，决定项目的取舍。

7.2.2　软件需求分析的目标

顺利地完成需求分析是一个艰巨的挑战。首先，要确认所有持有关键信息的人；然后需要从这些人那里获得可用的信息；最后才是把这些信息转化为清晰的和完整的形式。同时分析者还要考虑到可能的限制。除此之外他们还要考虑一个项目是否可行、是否在规定的时间里可以完成、价格上是否负担得起、是否合法、是否符合道德，一个新项目开始时人们往往还非常兴奋，往往试图轻视需求分析的必要性。但对过去项目的分析证明一个彻底的和无情的需求分析可以降低一个项目的耗费和降低其技术风险。

软件需求分析的目标是对软件的功能和性能进行描述，明确软件设计的约束和软件同其他系统元素的接口细节，定义软件的其他有效性需求。

需求分析阶段研究的对象是软件项目的用户要求。一方面，必须全面理解用户的各项要求，但又不能全盘接受所有的要求；另一方面，要准确地表达被接受的用户要求。只有经过确切描述的软件需求才能成为软件设计的基础。

通常软件开发项目是要实现目标系统的物理模型。作为目标系统的参考，需求分析的任务就是借助于当前系统的逻辑模型导出目标系统的逻辑模型，解决目标系统“做什么”的问题。其实现步骤如图 7-4 所示。

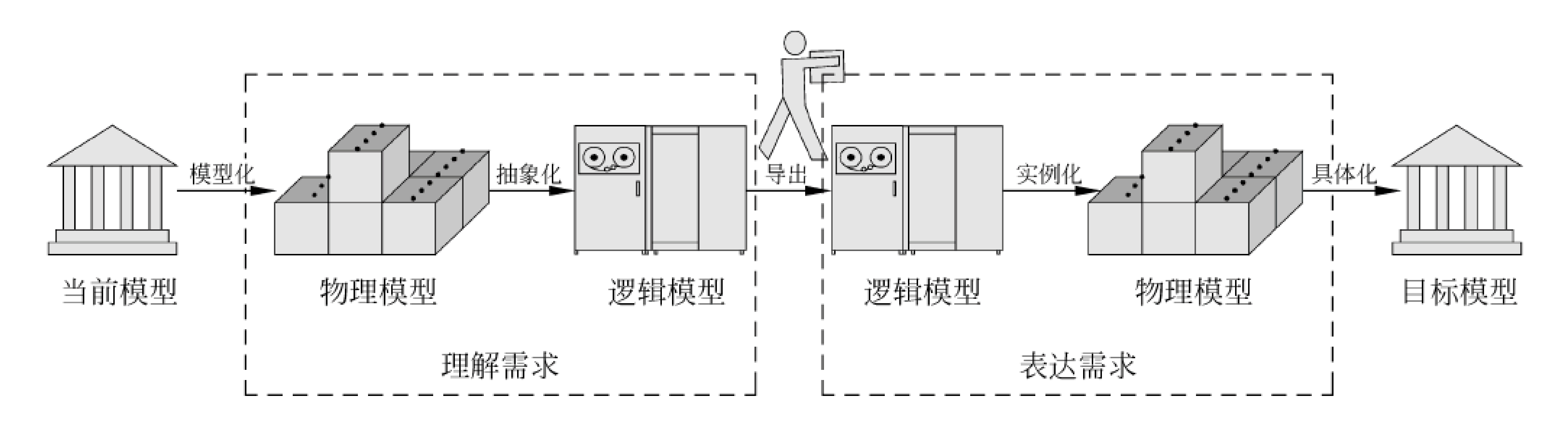


图 7-4　参考当前系统建立目标系统模型

7.2.3　需求分析的过程

需求分析阶段的工作，可以分成以下 4 个方面。

（1）问题识别

首先系统分析人员要确定对目标系统的综合要求，即软件的需求，并提出这些需求的实

现条件,以及需求应达到的标准。这些需求包括功能需求、性能需求、环境需求、可靠性需求、安全保密要求、用户界面需求、资源使用需求、软件成本消耗与开发进度需求,并预先估计以后系统可能达到的目标。此外,还需要注意其他非功能性的需求,如针对采用某种开发模式,确定质量控制标准、里程碑和评审验收标准、各种质量要求的优先级等,以及可维护性方面的需求。

此外,要建立分析所需要的通信途径,以保证能顺利地对问题进行分析。分析所需的通信途径如图 7-5 所示。

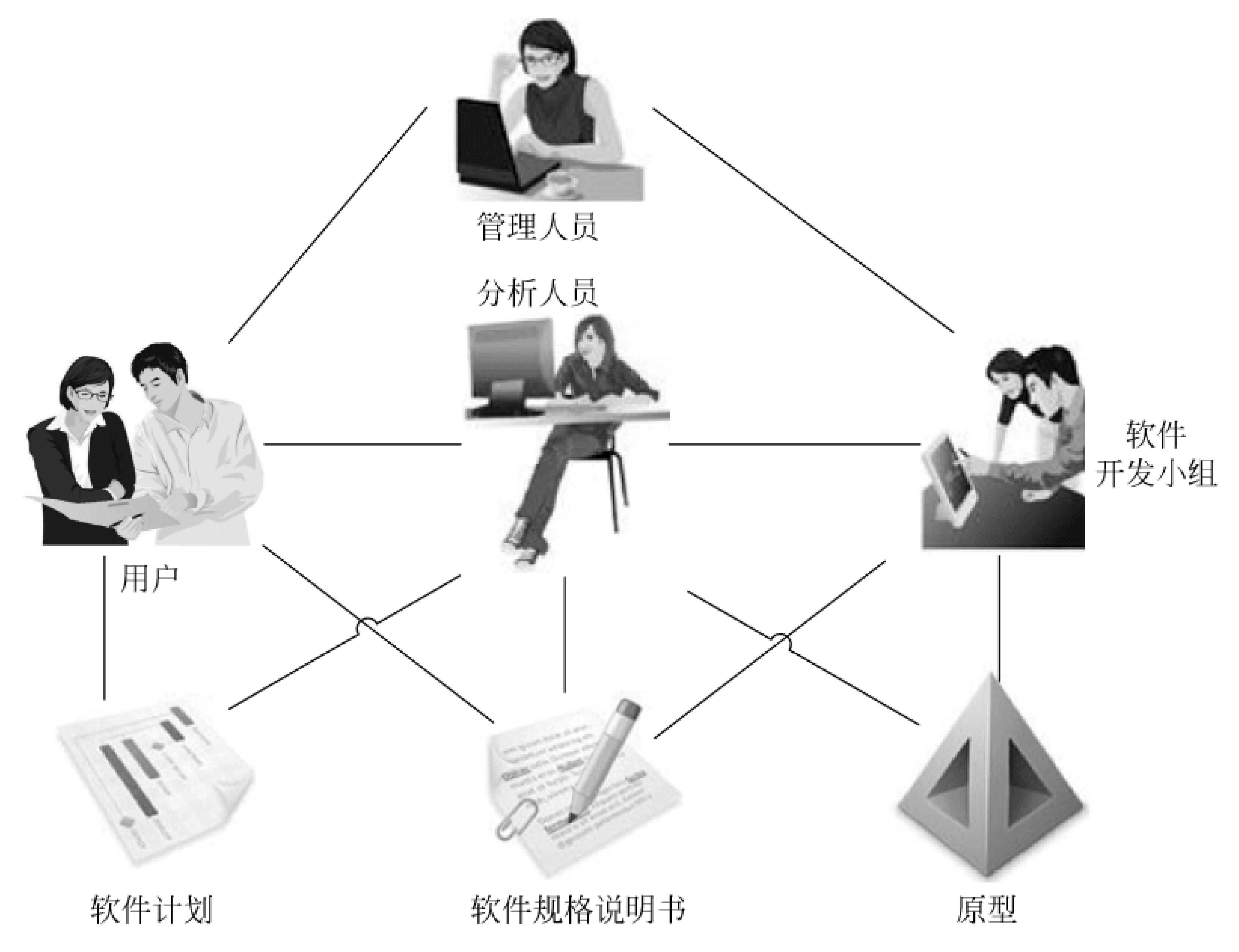


图 7-5 软件需求分析的通信途径

(2) 分析与综合

问题分析和方案的综合是需求分析的第二个方面的工作。分析员必须从信息流和信息结构出发,逐步细化所有的软件功能,找出系统各元素之间的联系、接口特性和设计上的限制,判断是否存在因片面性或短期行为而导致的不合理的用户要求,是否有用户尚未提出的真正有价值的潜在要求。剔除其不合理的部分,增加其需要部分,最终综合成系统的解决方案,给出目标系统的详细逻辑模型。

(3) 编制需求分析阶段的文档

已经确定下来的需求应当得到清晰准确的描述。通常把描述需求的文档叫做软件需求说明书。同时,为了确切表达用户对软件的输入输出要求,还需要制定数据要求说明书及编写初步的用户手册。

(4) 需求分析评审

作为需求分析阶段工作的复查手段,应该对功能的正确性、文档的一致性、完备性、准确性和清晰性,以及其他需求给予评价。为保证软件需求定义的质量,评审应以专门指定的人员负责,并按规程严格进行。评审结束应有评审负责人的结论意见及签字。除分析员之外,用户/需求者,开发部门的管理者,软件设计、实现、测试的人员都应当参加评审工作。

7.2.4　需求获取技术

需求获取技术包括以下两方面的工作。

(1) 建立获取用户需求的方法的框架。

(2) 支持和监控需求获取的过程的机制。

为了获得用户需求,可以通过以下途径进行调查研究。

(1) 了解系统的需求。软件开发常常是系统开发的一部分,仔细分析研究系统的需求规格说明,对软件的需求获取是很有必要的。

(2) 市场调查。了解市场对待开发软件有什么样的要求;了解市场上有无与待开发软件类似的系统。如果有,在功能上、性能上、价格上情况如何。

(3) 访问用户和用户领域的专家。把从用户那里得到的信息作为重要的原始资料进行分析;访问用户领域的专家所得到的信息将有助于对用户需求的理解。

(4) 考察现场。了解用户实际的操作环境、操作过程和操作要求。对照用户提交的问题陈述,对用户需求可以有更全面、更细致的认识。

在做调查研究时,可以采取如下的调查方式。

(1) 制定调查提纲,向不同层次的用户发调查表。

(2) 按用户的不同层次,分别召开调查会,了解用户对待开发系统的想法和建议。

(3) 向用户领域的专家或在关键岗位上工作的人个别咨询。

(4) 实地考察,跟踪现场业务流程。

(5) 查阅与待开发系统有关的资料。

(6) 使用各种调查工具,如数据流图、任务分解图、网络图等。

为了能够有效地获取和理清用户需求,应当打破用户(需方)和开发者(供方)的界限,共同组成一个联合小组,发挥各自的长处,协同工作。

为了更好地讲解随后出现的软件工程知识点,这里引出一个ATM自动柜员机例子,作为贯穿本书的一个案例。

一个实例:自动取款机系统(ATM)项目需求

(1) 项目背景

我们这个社会已经是信息社会,如果要花上很多时间去银行营业厅去排队等候办理取款转账等业务,对于那些每天生活节奏紧张的人来说,是一件不愿意做的事情。为了解决上述问题,建设一个可以24小时服务的自动取款系统无疑是最佳的。随着我国社会经济的快速发展,自动柜员机在全国金融界的不断普及给大众带来极其方便的自助交易业务。自动柜员机不但可以减轻营业网点压力、减少排队现象及人工出错率,同时也成为各家银行增长量较快的业务,给银行带来了巨大的经济效益。

(2) 所面向的客户

自动柜员机的用户是银行的合法顾客,使用自动柜员机的用户必须持有有效的银行卡,并且能够读懂操作说明。(这里的有效的银行卡指的是属于银联并未挂失的银行卡。)

(3) 系统总体需求

① 自动柜员机能识别出客户所插入的磁卡的类别,如果该磁卡不是有效的磁卡则执行退卡操作。

② 客户查询自己账户时,能够显示账户余额,并且根据输入的日期查询账户交易明细。

③ 客户在提取现金时,提示客户输入取款金额,并且判断所输入的金额是否正确,如果所输入的金额是错误的,则提醒客户并要求重新操作,如果所输入的金额是正确的,则提醒客户收取现金。

④ 客户在转账汇款时,提示客户选择转账类型,并且要求用户输入转账账号两次,以确保没有输入错误的转账账号,在两次输入都正确的情况下,让用户输入转账金额,并且系统给出界面,让客户做最后的确认。

⑤ 客户在进入修改密码界面后,要求用户输入新密码,并要求再次输入以确保密码统一无误,在客户确认后完成修改操作,并提醒客户新密码生效。

⑥ 照顾到客户情绪,系统的响应时间必须在客户可以承受的时间范围内响应用户的请求,并能够针对客户相应的请求,系统执行正确的操作。

⑦ 系统能够对错误的操作或者操作延时做出有效的处理。

(4) 系统要求

① 系统操作要求:

客户每次的取款金额小于等于 1000 元。

客户每天的取款金额小于等于 5000 元。

客户每次的取款金额必须是 50 的整数倍。

客户连续输入错误密码次数不能超过 3 次。

② 系统性能要求:

客户每次操作的响应时间不能大于 10 秒钟。

(5) 系统设计目标

ATM 自动取款机系统操作简单易用,能够为客户提供取款、转账/汇款、查询账户信息等服务,能够实现 24 小时不间断服务。

7.3 结构化分析方法

需求分析的方法有很多,这里介绍结构化分析方法(简称 SA 方法)。

结构化分析是 20 世纪 70 年代末,由 Demarco 等人提出的,旨在减少分析活动中的错误,建立满足用户需求的系统逻辑模型。该方法的要点是:面对数据流的分解和抽象,把复杂问题自顶向下逐层分解,经过一系列分解和抽象,到底层的就都是很容易描述并实现的问题了。

结构化分析是一种活动,是一种创建模型的活动。所建立的模型要达到 3 个基本目标:①能够准确地描述出用户的需求;②能够为后续的设计工作奠定一个良好的基础;③一旦

开发出软件产品之后，就可以用定义出来的需求作为标准，进行验收①。

结构化分析使用数据流图、数据字典、实体-关系(E-R)图、状态图等工具，来建立一种称为结构化说明书的目标文档——需求规格说明书②。利用这些工具，在结构化分析过程中导出的分析模型的形式，如图 7-6 所示。



图 7-6 分析模型的形式

分析模型以"数据字典"为核心，它是系统中各类数据描述的集合，是进行详细的数据收集和数据分析所获得的主要成果。它描述软件使用或产生的所有数据对象。在这个核心的周围，有 3 种不同的图，它们是数据流图、实体-关系图、状态图。

数据流图就是采用图形方式来表达系统的逻辑功能、数据在系统内部的逻辑流向和逻辑变换过程，是结构化系统分析方法的主要表达工具及用于表示软件模型的一种图示方法。建立数据流图的目的在于：描绘出信息在软件系统中流动时，数据怎样被变换；描绘变换数据流的功能和子功能。数据流图是功能建模的基础。在处理规格说明中给出了对出现在数据流图中的每个功能的描述。

实体-关系(E-R)图是指以实体、关系、属性 3 个基本概念概括数据的基本结构，从而描述静态数据结构的概念模式。E-R 图是用来进行数据建模活动的图形，图中出现的每个数据对象的属性可以在数据对象描述中描述。

状态图指明了作为外部事件结果的系统行为，它描绘了系统的各种状态(即行为模式)和在不同状态间转换的方式。状态图可以作为行为建模的基础。有关软件控制的附加信息可以在控制规格说明中描述。

7.3.1 数据流图

数据流图(Data Flow Diagram，DFD)是用于表示软件系统逻辑模型的一种图形，一个基于计算机的信息处理系统由数据流和一系列的转换构成，这些转换将输入数据流变换为输出数据流。数据流图反映客观现实问题的工作过程。它用简单的图形记号分别表示数据流、加工、数据存储以及外部实体。数据流图中没有任何具体的物理元素，只是描述数据在系统中流动和处理的情况，具有直观、形象、容易理解的优点③。

1. 数据流图的组成部件

- 数据流用箭头──►表示
- 加工用圆◯表示

---

① 张海藩. 软件工程. 第 2 版. 2006

② 需求规格说明书编制规范参见《计算机软件文档编制规范(GB/T 8567—2006)》(该编制规范由中华人民共和国国家质量监督检验检疫总局和中国国家标准化管理委员会共同发布，其国家标准查询页面为 http://220.181.176.160/stdlinfo/servlet/com.sac.sacQuery.GjbzcxDetailServlet? std_code=GB/T%208567-2006，该编制规范具体内容参考 http://www.docin.com/p-6859638.html)

③ 肖汉. 软件工程理论与实践. 北京：科学出版社，2006

• 数据存储用双横线______表示

• 数据流的外部实体用长方形▭表示

（1）数据流

由一组固定成分的数据组成。如数据流“取款金额”由“取款金额大小”、“币种”等成分组成。数据流的作用就是反映数据信息的流动方向，它的流动方向一般有如下4种情况。

• 加工——►加工
• 加工◄—►数据存储
• 外部实体——►加工
• 加工——►外部实体

值得注意的是，可以同时有几股数据流，但是这几股数据流两两之间无任何联系，并且不是同时流出，否则归为同一股数据流。对于数据流的命名，一般从数据流组成成分或实际具体含义角度给每个数据流命名。另外，由于数据存储可以说明问题，所以与数据存储发生数据的流入或流出的数据流不必命名，图7-7是某个企业的商品订单处理的数据流图[①]，图7-8是学员报名的数据流图[②]。

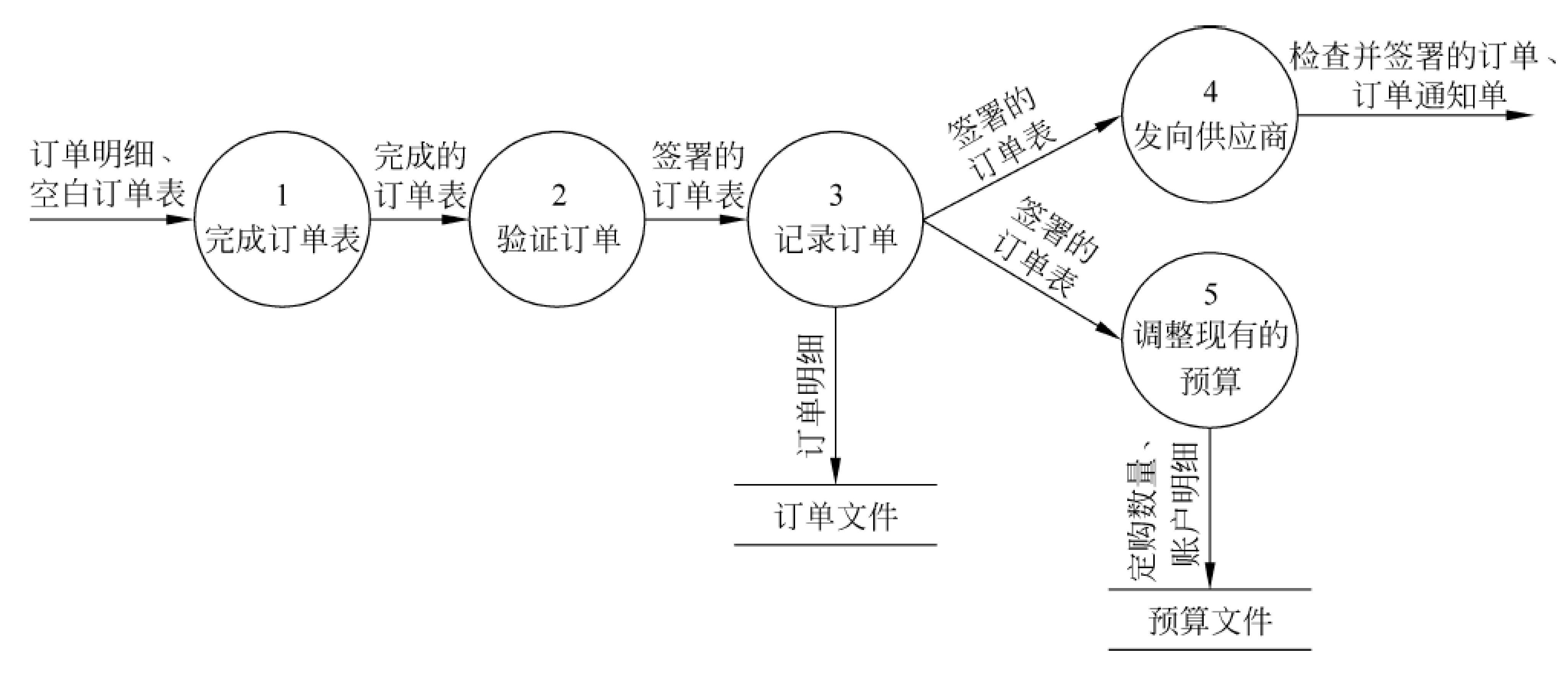


图7-7 商品订单处理的数据流图

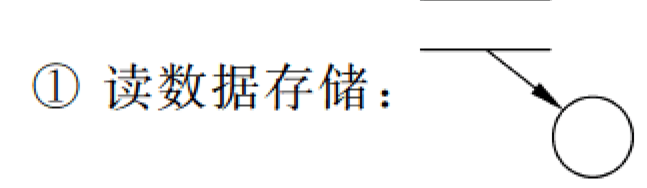
（2）加工

加工用于反映对数据进行某种操作，其命名采用用户习惯的且反映加工含义的名字，并加上编号（说明这个加工在层次分解中的位置）。一般来说，加工由一个具体的及物动词加上一个具体的宾语构成。例如，在图7-7中“完成订单表”、“验证订单”、“记录订单”等都是加工。

（3）数据存储

数据存储指暂时保存数据。它的命名应当适当选择，以便于理解。加工与数据存储之间的数据流向有如下3种。

① 读数据存储：



① Ian Sommerville. 软件工程. 北京：机械工业出版社，2003

② 师素娟. 软件工程教程. 河南：黄河水利出版社，1999

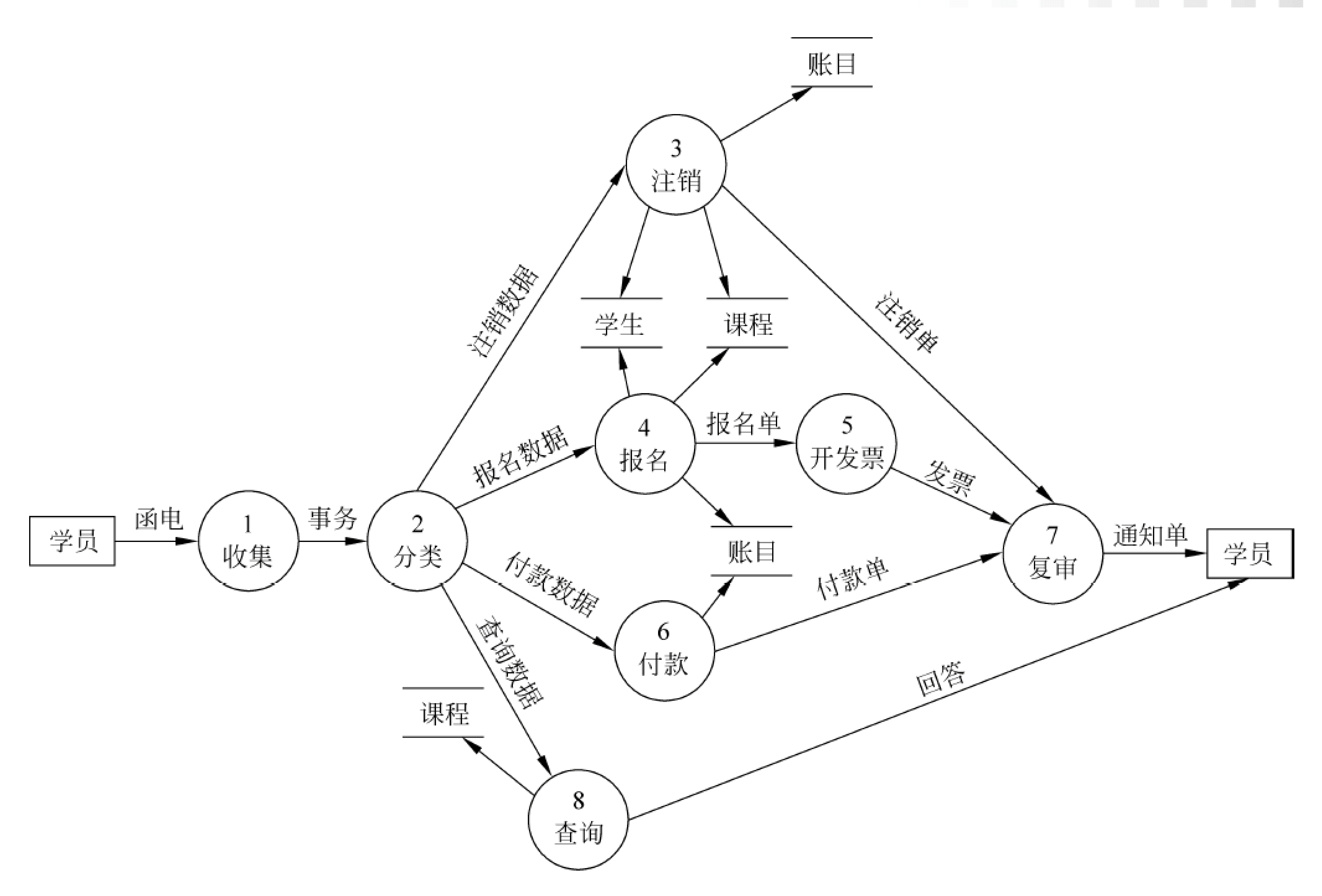
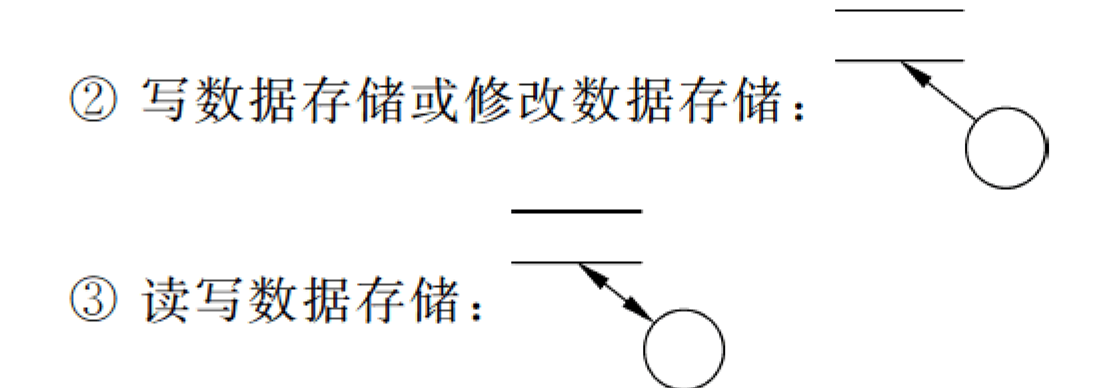


图 7-8　学员报名的数据流图



② 写数据存储或修改数据存储：

③ 读写数据存储：

(4) 外部实体

为了便于理解，有时可以画出数据流的外部实体来反映数据的来源与归宿，如图 7-8 中“学员”是数据流“函电”的源点，也是数据流“通知单”的终点。源点与终点通常是存在于系统之外的人员或组织，画出源点和终点只是起到注释作用帮助理解而已，它们是系统之外的事物，表达不必很严格。

2. 画数据流图的思考顺序

如何着手开始画数据流图？在需求分析初期只需将现实的状况反映出来即可，不用也无法一下子将系统的内部都详细地想象出来。我们首先考虑外部的环境和与系统交互的对象，然后再逐步向内深入，这种“由外及内”的思想是比较自然而且有条理的思考过程。具体来说，就是首先应画出系统的输入数据流和输出数据流，然后再考虑系统的内部；每一个加工也是先画其输入输出，再考虑其内部。

在画某个系统的数据流图时，第一个问题就是如何画系统的输入输出？

在需求分析的开始阶段，系统分析人员还无法清楚地了解整个系统都有哪些功能，这时要向客户了解：系统需要从外界获得什么样的数据？系统向外界送出什么样的数据？这两个主要问题此时要详细记录，并且分析客户对这两个问题的回答，从而分析出系统的输入和

输出。自动柜员机系统数据流图的外围如图 7-9 所示。

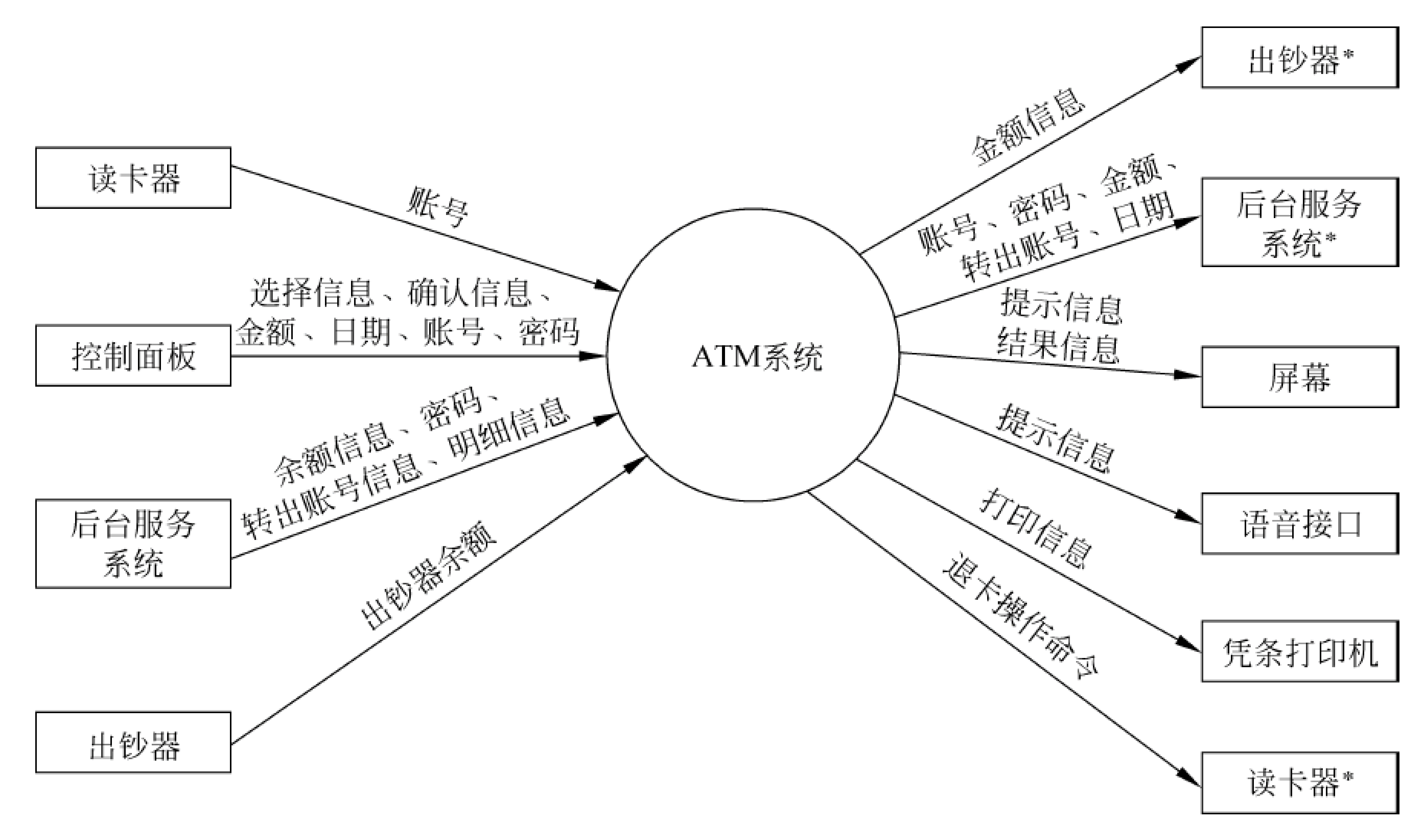


图 7-9 ATM 系统 0 层数据流图

第二个问题就是如何画系统的内部？这时需要设计一些加工，以及一些连接这些加工的数据流。设计加工和数据流的顺序一般是从输入端到输出端，也可以从输出端追溯到输入端。那么在哪里设计一个加工呢？加工应处于数据流的组成或值发生变化的地方。不管从源头加工还是终点加工来看，都需要了解数据流的组成。设计一个数据流需要了解它的组成是什么，这些组成项来自何处，这些组成项如何组合成这一数据流，为实现这一组合还需要什么有关的加工和数据等。除此之外，还要设计或标记出存储各种数据的数据存储，应该了解数据存储的组成情况。对自动柜员机系统来说，在图 7-9 的基础上，按照"由外及内"的思考顺序进行分析，就可以画出如图 7-10 所示的数据流图。

如果经过进一步分析，还有一些数据流在加工的内部，这样一个加工可以分解为几个子加工，这几个子加工之间再用数据流连接起来。先为数据流命名，再为加工命名。画数据流图时，应该要抓住主要矛盾：把重点放在主要的数据流上。暂时先不考虑一些例外情况、出错处理等枝节性问题，只表示出这种数据流即可。系统分析人员要做好随时更正甚至抛弃原来绘制的数据流图的准备，因为理解一个事物总是需要一个过程，这符合事物的发展观念，理解需求总是要从不正确到正确，从不恰当到恰当，如果是一个大型的系统，更是如此。

3. 数据流图的画法

上面说到了有些加工可以分解，如果是一个大型的软件系统，用一张数据流图来画出一个所有的数据流和加工，那么整个画面太过于庞大和复杂，实际上不利于对系统的理解。为了克服这个问题，可以考虑使用"自顶向下逐层分解"的结构化分析方法。这种方法将大问题分割成中问题，中问题分割成小问题，这体现了分解和抽象的精神。具体来说，就是将数据流图分层描绘，一般由顶层、底层、中间层组成。

顶层数据流图是系统的边界，称为 0 层，以 ATM 系统为例，其顶层数据流图如图 7-9 所示。

中间层处于底层和顶层之间，描述了某个加工的分解，而它的组成部分又要进一步分解。以 ATM 系统为例，其中间层数据流图如图 7-10～图 7-13 所示。

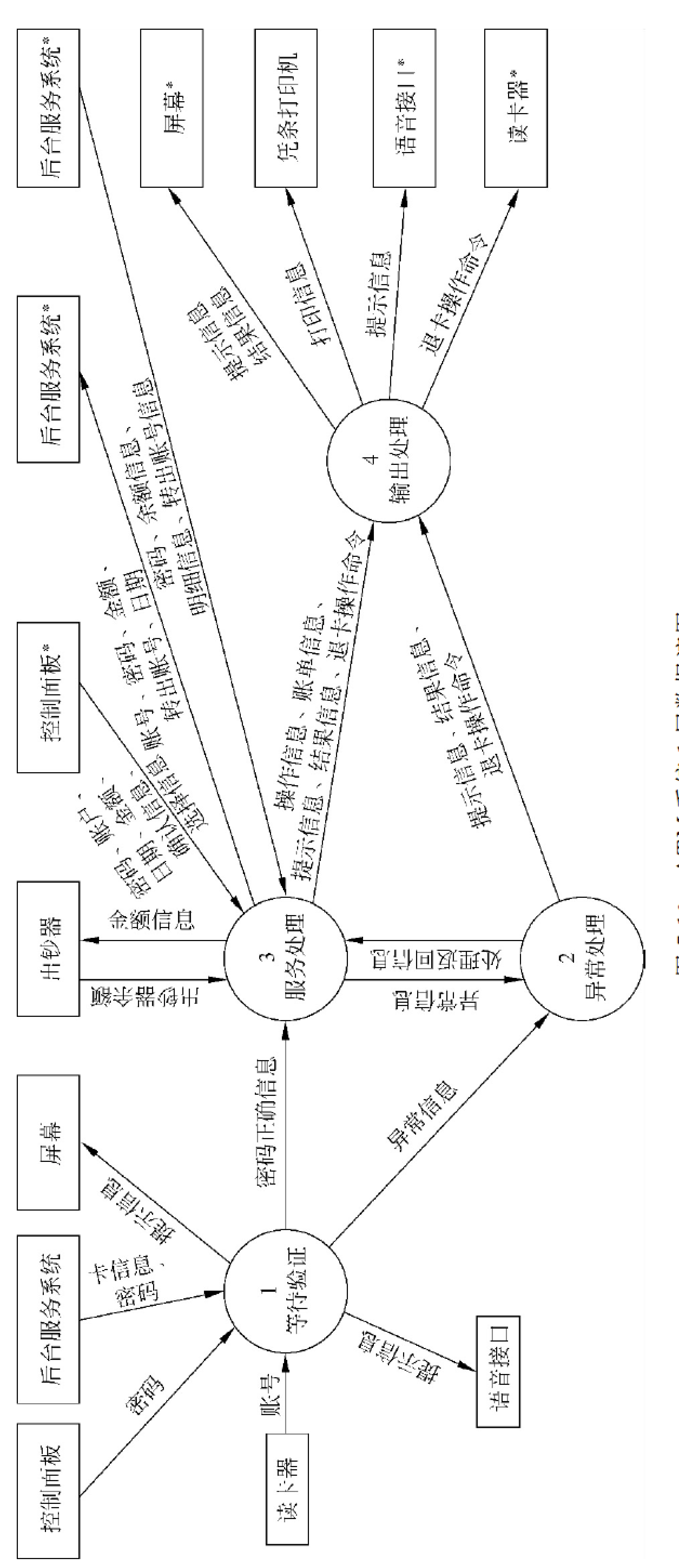


图 7-10 ATM 系统 1 层数据流图

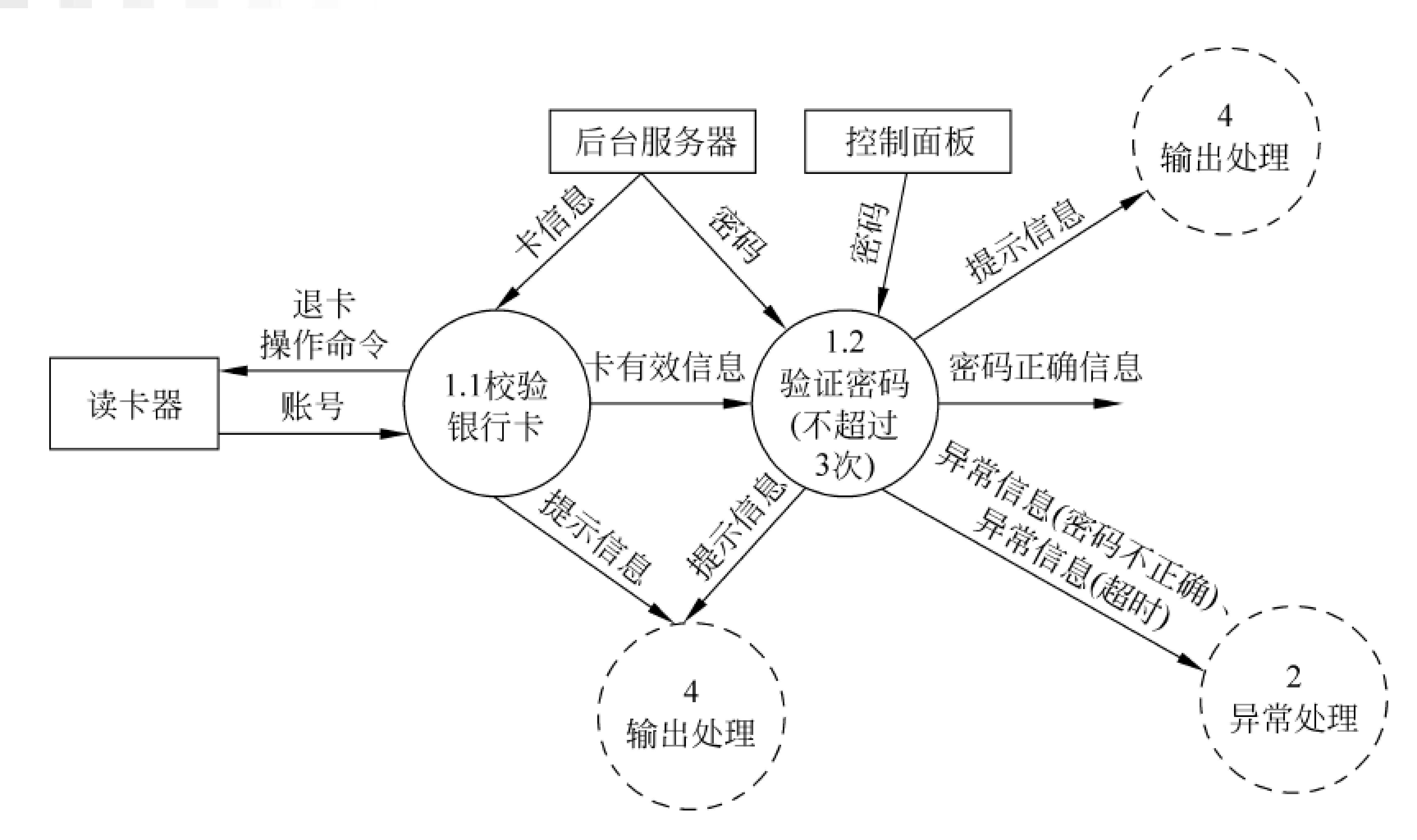


图 7-11 ATM 系统 2 层数据流图(“1—等待验证”展开)

余额信息、确认信息、
出钞器余额
处理返回信息
(重新输入金额)
异常信息
(金额出错、超时)
金额信息、提示信息、返回信息
操作信息、账单信息
3.3
取款处理
余额信息、账户、
确认信息、转出账号信息
提示信息、金额信息、转出账号、返回信息
操作信息、账单信息
3.4
转账处理
异常信息(卡号出错、金额出错、超时)
处理返回信息
(重新输入卡号、金额)
选择取款的信息
选择转账
的信息
密码正确信息
3.1
选择服务
服务类型
3.2
判断服务
类型
余额查询
的信息
3.5
余额查询
处理
账户、返回信息、结果信息
异常信息(超时)
余额信息
明细查询
的信息
3.6
明细查询
处理
结果信息、
返回信息
异常信息(超时)
明细信息、日期
异常信息(超时)
退卡操作命令
选择修改密码的
信息
3.7
修改密码
密码、确认信息
结果信息、提示信息、
密码、返回信息
异常信息
(新密码不一致、超时)
处理返回信息
(重新输入新密码)

图 7-12 ATM 系统 2 层数据流图(“3—服务处理”展开)

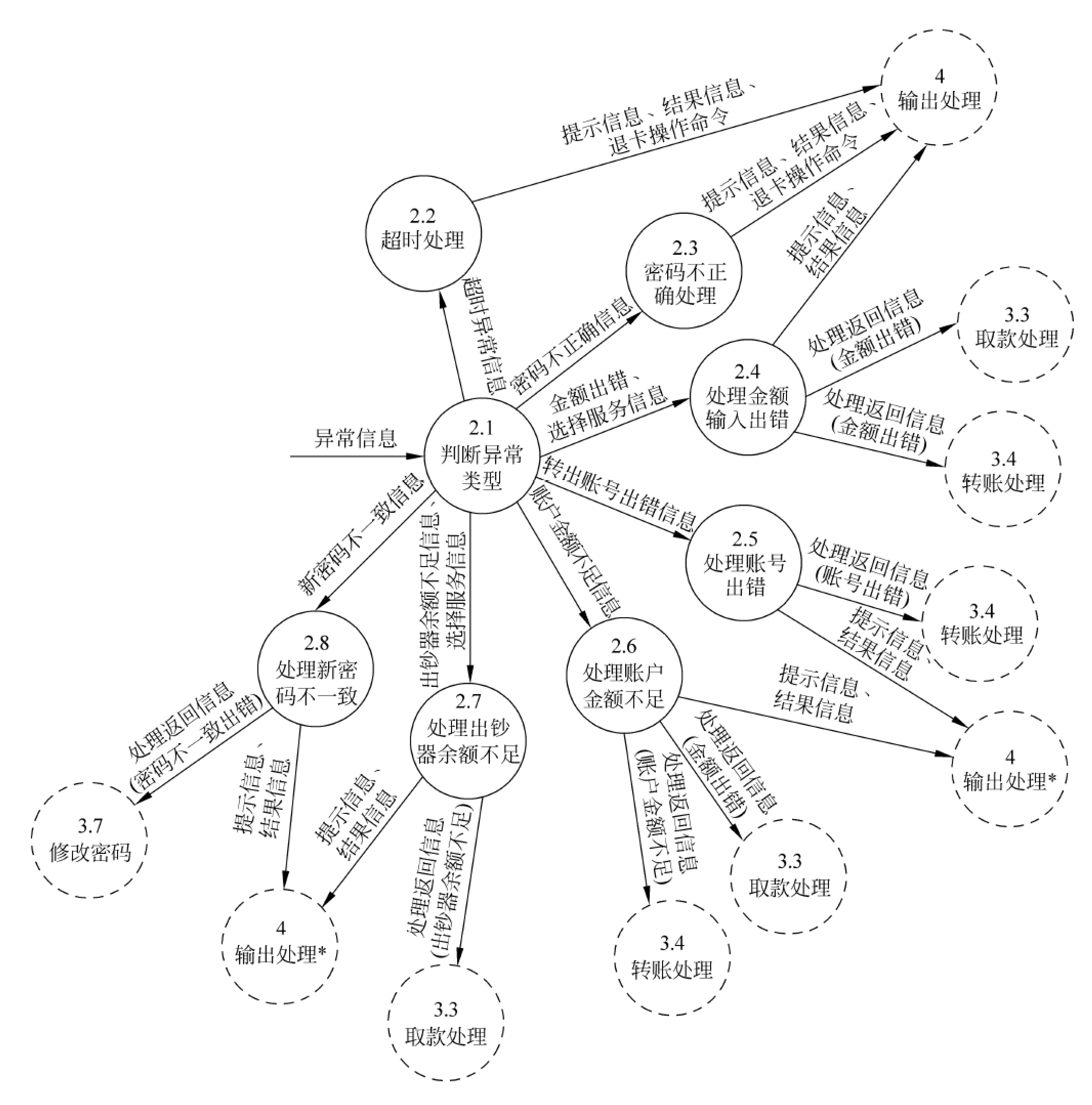


图 7-13 ATM 系统 2 层数据流图(“2—异常处理”展开)

底层称为基本加工,由一些不再分解、足够简单的加工组成,以 ATM 系统为例,其底层数据流图如图 7-14～图 7-17 所示。

一般来说,越大的系统中间层越多。在分层的数据流图中,父图中有几个加工,它就可以有几个子图。其顶层称为 0 层,往下是 1,2,3,…层,0 层是第 1 层的父图,而第 1 层既是 0 层图的子图,又是第 2 层图的父图,依次类推①。

在绘制 SA 方法分层数据流图时,应注意以下一些要点。

既然是分层,对各层之间的加工和数据流如何标识,对图与图之间如何标识,以表明父子层次关系?答案是:编号。为了能够更好地理解和管理,可以考虑按以下规则为数据流图和其中的加工编号:①子图的编号即子图号就是父图中相应加工的编号;②子图中加工的

① 张雅军.浅析软件工程中的数据流图的画法[J].天津职业院校联合学报,2008,10(2):70～73

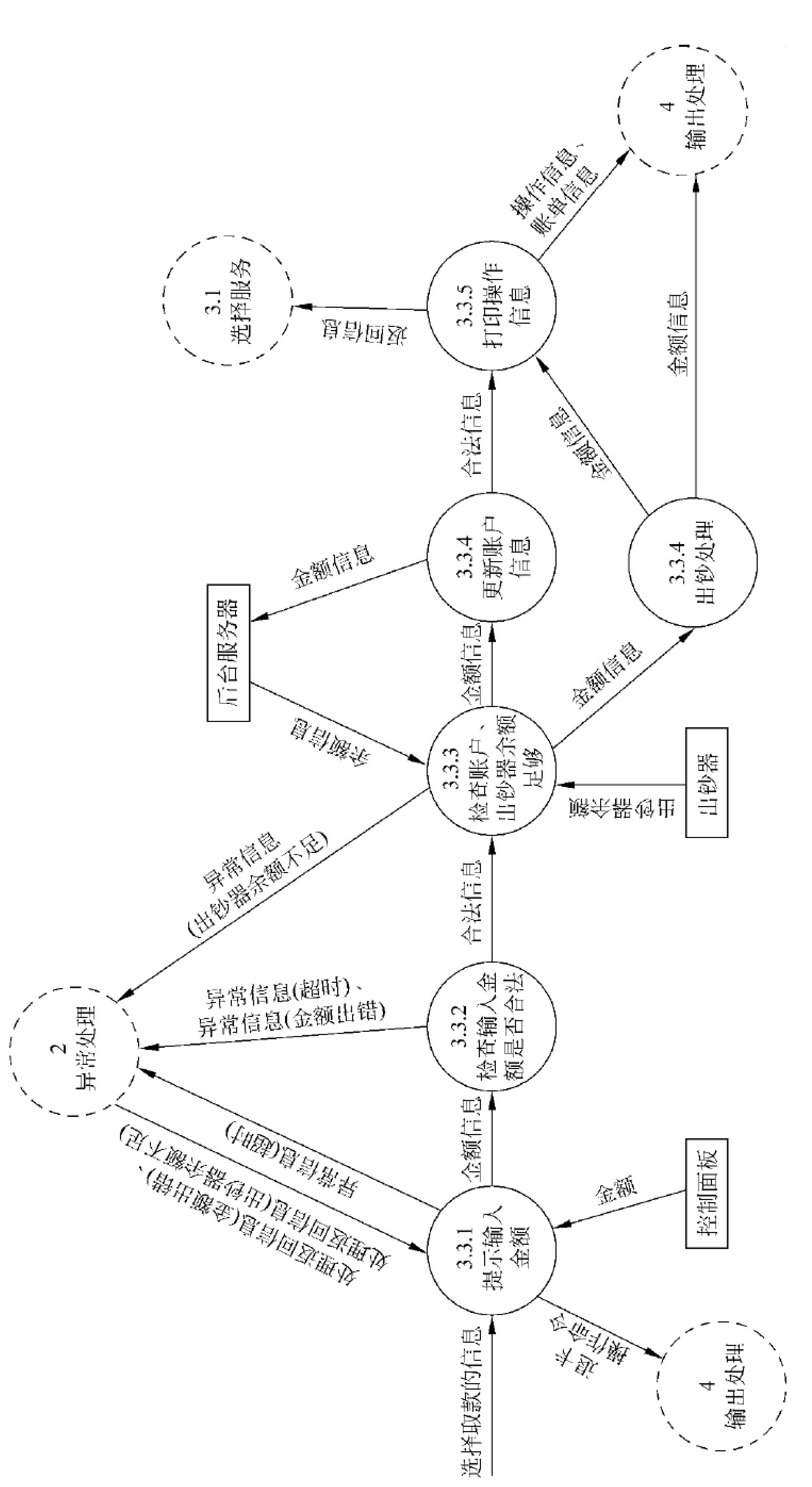


图 7-14 ATM 系统 3 层 DFD(“3.3—取款处理”展开)

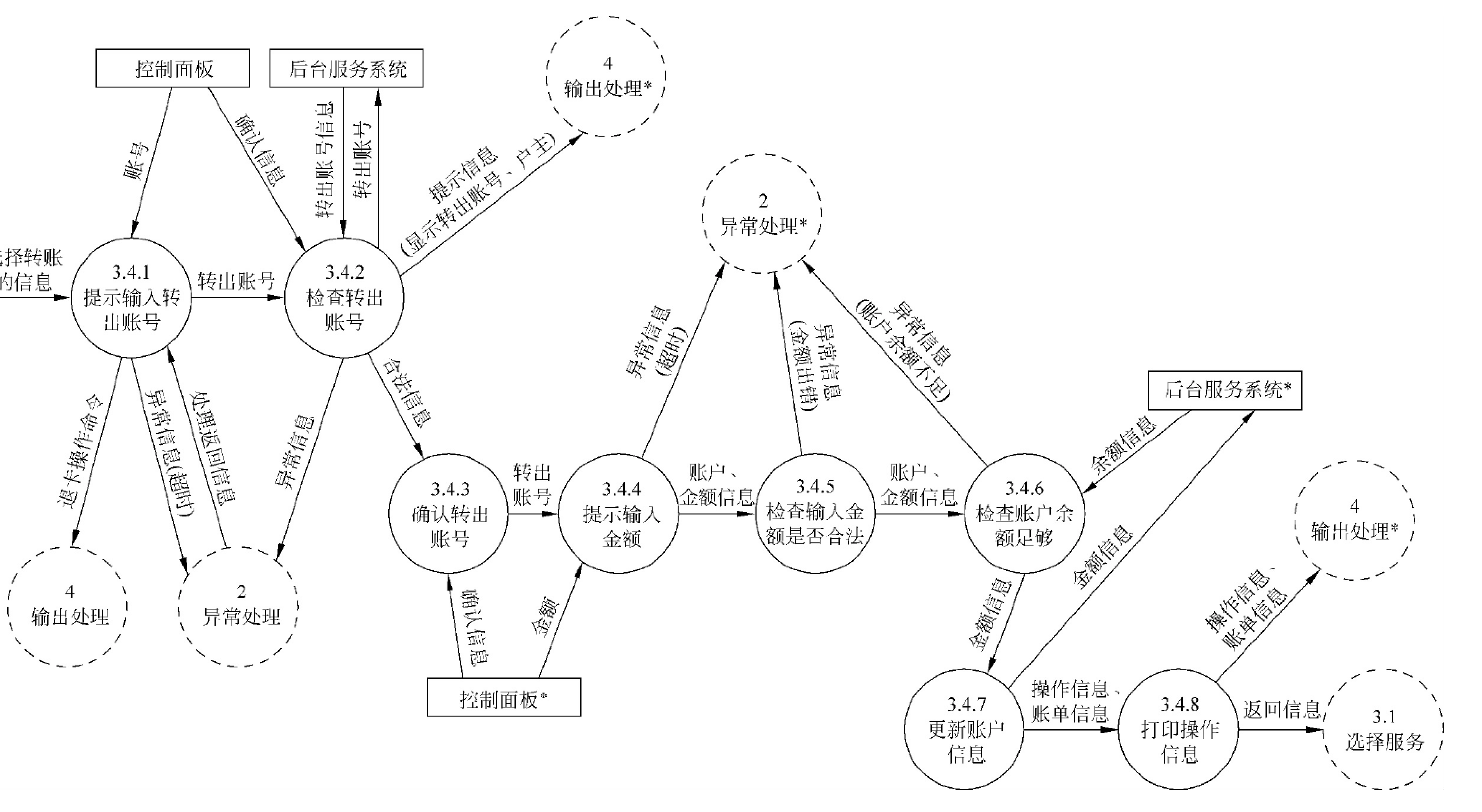


图 7-15　ATM 系统 3 层 DFD(“3.4—转账处理”展开)

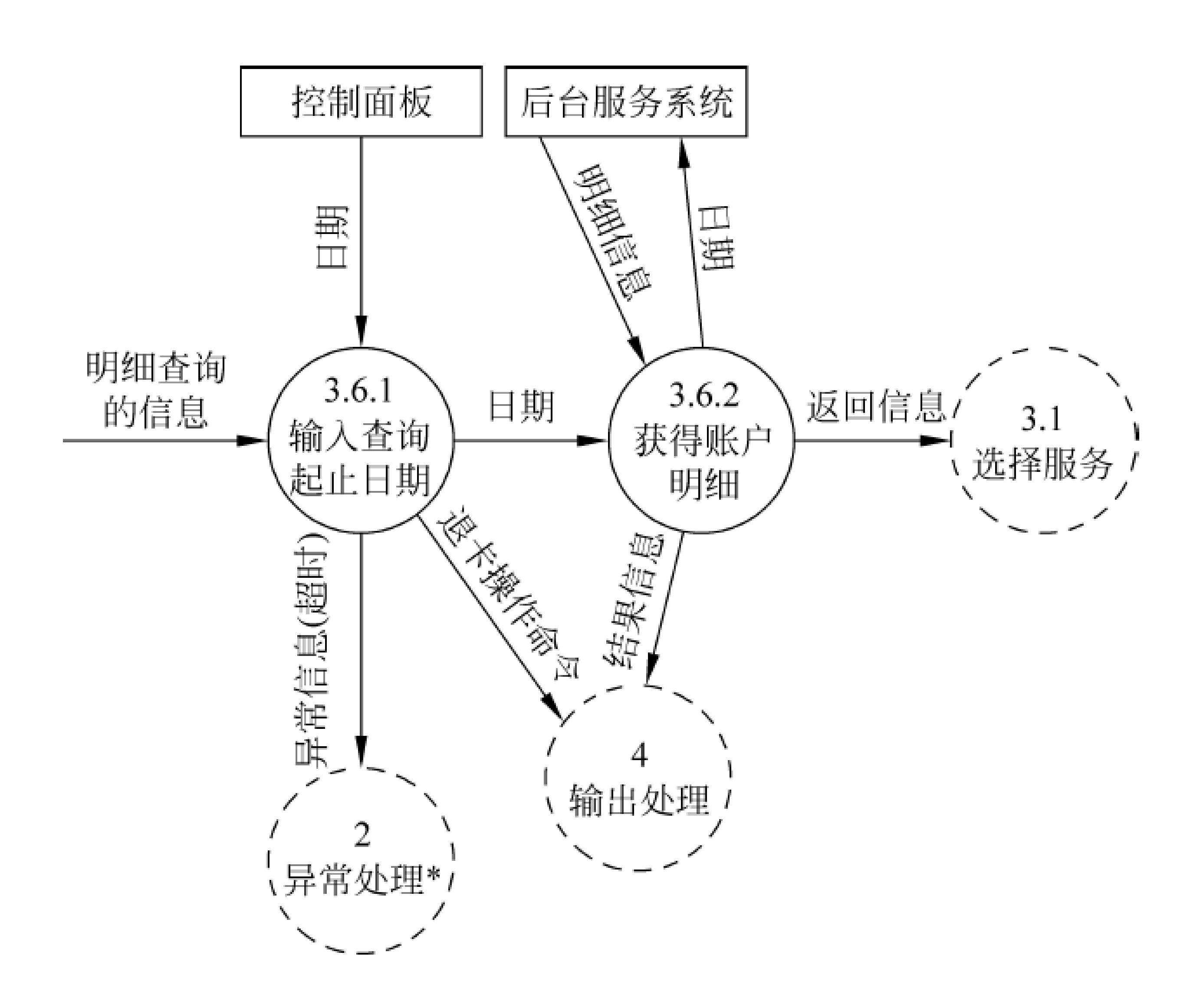


图 7-16 ATM 系统 3 层 DFD(“3.6—明细查询处理”展开)

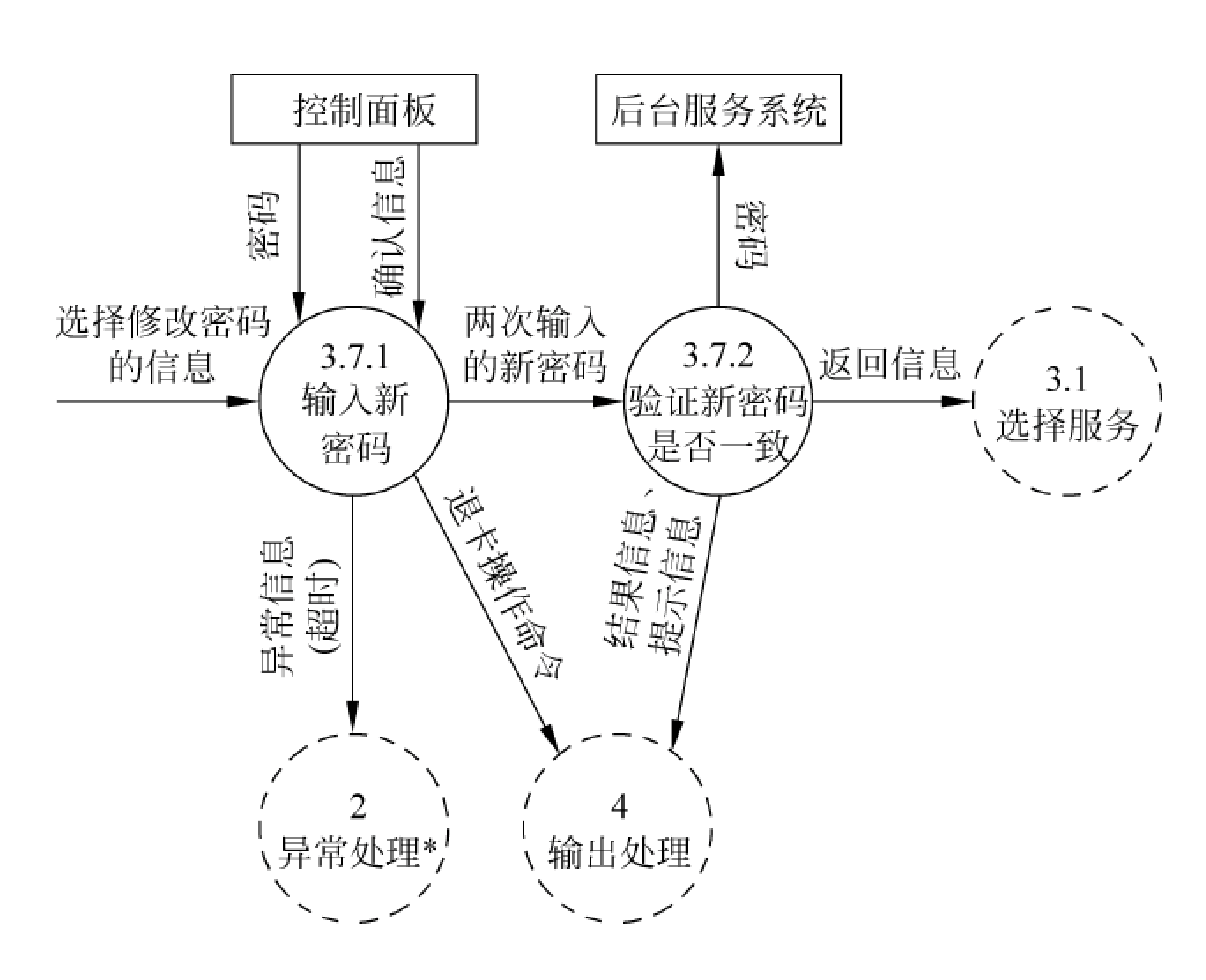


图 7-17 ATM 系统 3 层 DFD(“3.7—修改密码”展开)

编号由子图号、小数点、局部顺序号连接而成。(注：顶层图不必编号,加工一般只有一个;其下一层编号为 0,图中加工的编号就是 0.1,0.2,0.3,通常删去小数点前面的 0,所以这些加工的编号就是 1,2,3。)图 7-10“ATM 系统 1 层数据流图”是图 7-11“ATM 系统 2 层数据流图”的父图。

按照分层的思想,就是要对数据流图进行不断的分解,那么分解到什么时候才能结束,我们在上面对底层数据流图定义做过解释,底层数据流图包含一些基本加工,由一些不再分解、足够简单的加工组成。在需求分析阶段,结构化分析的最终目的是要分解到只包含基本加工的数据流图。还有一个问题,即中间层需要多少层次?如果层次过多,会给理解带来困

难。需求分析实际上是技术加艺术的过程，优秀的分解会使系统更利于理解，同时也使后续的设计工作更加顺畅，我们在分解时可以遵循这几个原则(但不限制)：①合理性，即分解在逻辑上应合理、自然，不能做硬性分割；②层次少，即在保证数据流易理解性的前提下，尽量使分解层次数少，也就是可以适当多分解成几部分；③分解均匀，即在一张数据流图中，不要有些加工已是基本加工，而另一些加工还要再分解好几层，如3～4层；④一个加工最多分解为7～8个子加工，这点与②有点互为矛盾，一般来讲层次少，那么每层分解的加工就多，反之亦然，所以这还是一个平衡问题，具体分析过程中要掌握好平衡度；⑤操作上，上层可分解得快一些，下层应分解得慢一些。

要勇于和善于对数据流图进行改进，系统分析人员对系统的理解，不可能一蹴而就，我们需要实践→认知→再实践的往复过程，画数据流图也需要这样一个过程，一开始画出来的数据流图可能不完整、不准确，就要对其进行再次检查，把不完整、不准确的地方纠正过来，甚至可能需要重新画过。表7-1是对数据流图进行验证的角度和要点。

表7-1 验证数据流图正确性的方法

验证数据流图正确性	要　　点
合理使用数据存储	在画数据流图时，需要注意加工与数据存储间数据流的方向。加工要读数据存储时，则数据流的箭头是指向加工的；加工要写数据存储时，则箭头是指向数据存储的；加工要修改数据存储，箭头也应指向数据存储而不是双向的。只有当加工除了修改数据存储之外，为了其他目的，还要读该数据存储时，才画双向箭头。应当注意，一般有写数据存储的加工同时也应有读数据存储的加工，否则就有某些加工漏画了。同时还应正确地画出加工与数据存储之间数据流的方向
保持父图和子图的平衡	在绘制SA方法分层数据流图时，还需要注意保持父图和子图的平衡。例如，图7-10“ATM系统1层数据流图”是父图，图7-12是它的一张子图。父图中的加工3被分解成子图中的7个加工。子图实际上就代表了父图中的加工3，它是对加工3的详细描述。在绘制子图时，其输入、输出数据流应该和父图中加工3的输入输出相同，这就是父图和子图的“平衡”问题。更具体地说，“平衡”是指在借助数据字典并可忽略枝节性数据流的情况下，父图中加工的输入输出，与其对应的数据流子图的所有输入输出数据流必须相同。父图和子图必须平衡，这是分层数据流图的重要性质，平衡的分层图是可读可理解的，反之数据流图就无法理解。图7-12是平衡的，因为父图中加工3的输入输出与子图中的输入输出完全相同。对于分层数据流图的“平衡”问题，一些学者提出了父子数据流图平衡的形式化定义[①]和平衡验证手段[②]
保持数据平衡	如果某个加工有输出但没有输入，或者一个加工用于产生输出的数据并没有相应的输入，则肯定有数据流漏画了；如果加工的某些输入没有从这个加工输出，则需要考虑这个数据流是否必要，不必要则将其去掉

① 秦晓. 数据流图的形式规范[J]. 软件学报，1994，5(5)：39～45
② 江志超，张家重，刘方爱，刘培玉. 数据流图的形式化研究[J]. 理论研究，1993，8：1～4

表 7-2 是提高数据流图的易读性的要点。

表 7-2 提高数据流图易读性的方法

提高数据流图易读性	要　　点
分解要均匀	有一种情况是这样的：在一张数据流图中，一些加工已是基本加工，而另一些加工却还可进一步分解成好几层，那么这张数据流图的易读性就不是很好，因为在同一张数据流图中某些部分描述的是底层细节，而另一些部分描述的却是较高层的抽象。要提高数据流图的可读性，是将一个问题分解成大小均匀的几个部分。要完全做到这点当然不容易，尤其是一个大型系统，只能尽量避免特别不均匀的分解。如果上面说到的这种分解很不均匀的情况，就需要考虑重新分解(对整个父图重新分解的步骤，如表 7-3 所示)
加工之间的联系要简化	要尽量减少加工之间的数据流的数量，这样可以使各个加工之间的耦合性(耦合性的概念在第 8 章中介绍)更加松散些，同时可以使各个加工的独立性更强，即加工的内聚性(内聚性的概念在第 8 章中介绍)更高些。合理的分解应是将一个问题分成相对独立的几个部分，这样每个部分就可单独去理解。既然要求各个加工的独立性强，就必须使它们之间的联系要少，不画多余的数据流线
命名要合理	要注意数据流图中各成分的命名要正确、准确、无二义，合理的命名可以提高数据流图的可读性

如果出现分解不均匀或加工的内聚性不好，或者是出现某些大问题时，如绘制某一加工的子图时，子图中存在相互独立的组成部分，这是不合理的，需要推翻整个父图，进行重新分解，如图 7-18 所示①。

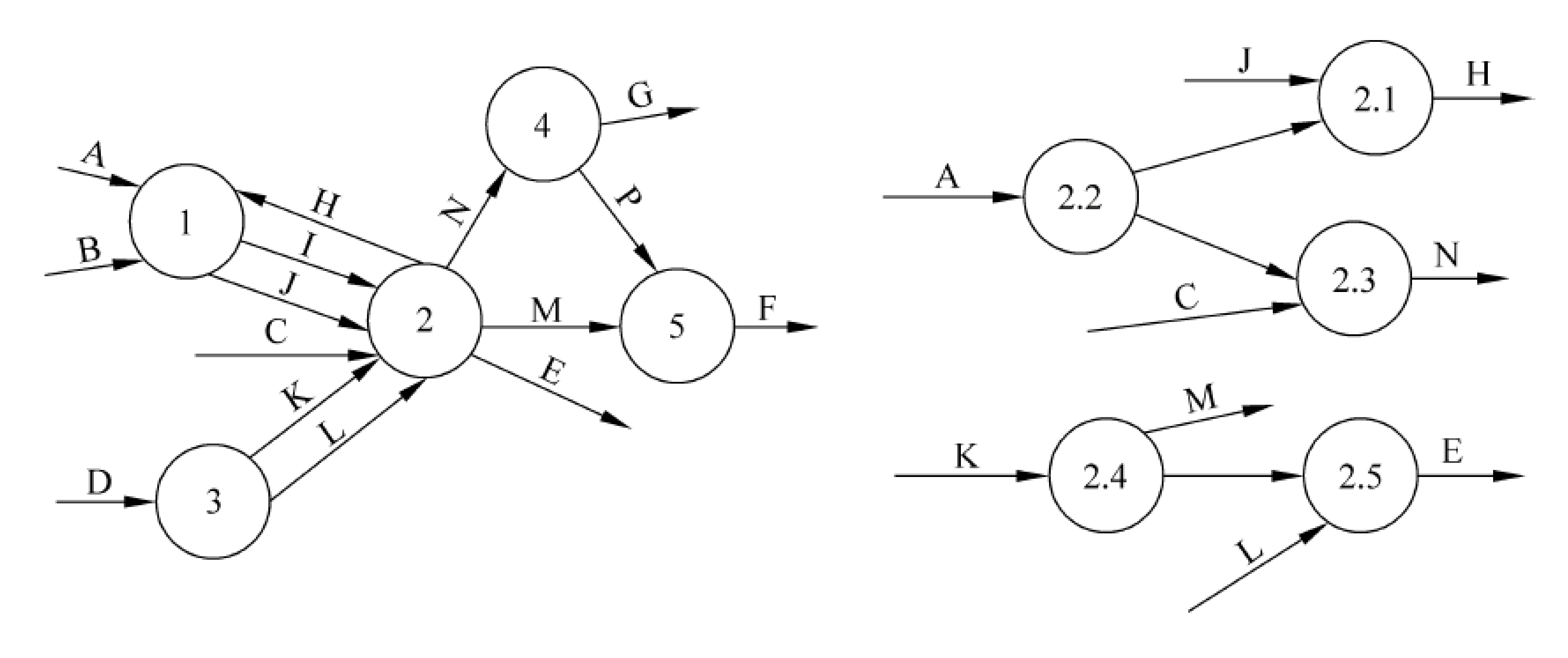


图 7-18 不合理的父子数据流图

如果出现要重新分解整个父图时，可以按表 7-3 所示步骤进行。

表 7-3 重新绘制父图的步骤

步骤序号	步 骤 内 容
1	重新绘制一张图，将需要重新分解的某父图的所有子图都放在一起
2	遵循提高独立性的原则，观察图形，将这张图切分成几部分，使各部分之间的联系最少。再观察图形，可以适当调整各组成部分的位置，以利于切分

① 肖汉. 软件工程理论与实践. 北京：科学出版社，2006

续表

步骤序号	步 骤 内 容
3	将第 2 步切分得到的每个部分以圆圈框住，即为父图的每个加工，各部分之间的联系就是父图加工之间的界面了，这样，就可以重建父图
4	父图重建完之后，就可以重建各张子图。因为在第 2 步中已将原子图的设计打乱，这时只需把第 2 步所得的图按各部分的边界剪开即可
5	为新的父图、子图中的所有加工重新命名和编号

4. 数据字典

数据流图由数据流、加工、数据存储、外部实体组成，描述了系统的“分解”，但其中的这些组成部分的含义并不知道，这时就需要用到数据字典。数据字典(Data Dictionary，DD)是一种表达数据元素的工具。软件需求分析图中出现的所有数据元素都必须在数据词典中给出逻辑定义。通常，数据字典与数据流图是配合使用的。数据流图中的组成部分的名字都是一些属性和内容的抽象概括，不同的人对这些组成部分的理解可能是不同的。对于需要团队开发的软件项目来说，这种不同的理解会给以后的开发和维护工作带来困难。数据字典是所有与软件系统相关的数据元素的一个有组织的列表，它给出了所有需要表达的组成部分的精确的、严格的定义，使用户和系统分析员对于输入、输出、存储成分和中间计算有一致的理解。

编写数据字典要尽可能严密精确，以下是数据字典的构造准则。

(1) 数据流图中出现的名字都应当成为数据字典中的一个条目。

(2) 数据流图中的数据存在着(数据元素)→(数据流、数据存储)的结构关系，这是一种从左至右的构造方式。即先定义数据元素，由数据元素定义数据流和数据存储。

(3) 不能重复定义。即一个条目只能对应一个名字，而一个名字只能有一个条目。

(4) 定义时所用的词汇都应具有明确的含义，即对该含义的理解无二义性。

5. 数据元素

在系统中，直接反映事物某一特征信息的元素称为数据元素，它是定义复杂数据的基石。一般来说，许多数据项较简单，它是事物某一特征的概括，这个名字应当有公认的明确定义，大家对它都只能有一种理解，不需要再定义，如年龄、性别、姓名、身份证号码等；但是有一些数据元素是说明信息，它具有一些特殊含义(如该数据元素的类型、值的范围、峰谷值等)，就需要定义，如成绩的表示可能是优、良、中、及格、不及格，也可能是 A、B、C、D、E、F，也可能是百分制 0～100，所以要定义清楚，如果是百分制 0～100，值的精度也要定义，即成绩精确到个位还是十分位。

例如，在上面的自动柜员机系统例子中的银行卡账号的描述如下。

数据元素名字：账号
别名：CardCode
简要说明：工商银行卡卡号为 19 位数字
类型：字符型
长度：19

取值范围及含义：
第 1～5 位：固定为 95588，全国相同
第 6 位："0"为牡丹灵通卡，"2"为牡丹灵通 e 时代卡，"8"为理财金卡
第 7～10 位：地区代码
第 11～19 位：编号

再例如，自动柜员机系统中取款金额的描述如下。

数据元素名字：取款金额
别名：GetAmount
简要说明：取款时输入的金额
类型：数字型
精度：20
小数位数：2
约束条件：
取款金额小于等于 5000
取款金额大于等于 50
取款金额必须为 50 的倍数

6. 加工规约

在结构化分析的需求分析阶段，还需要用加工规约对数据流图中的加工进行说明。对加工描述清楚，才能将软件系统的功能表述清楚。那是不是所有的加工都要有加工规约呢？因为数据流父图中加工规约就是对应数据流子图中各个基本加工说明的总和，所以只要对数据流图中的基本加工用加工规约进行说明即可。基本加工规约，不必描述具体用什么技术来实现加工的细节（这是个"怎么做"的问题），而主要是要描述加工"做什么"。目前，描述加工规约的工具主要有结构化语言、判定表、判定树。这里主要介绍结构化语言，判定表、判定树可以参见其他资料[1]。

结构化语言就是将自然语言加上程序设计语言的控制结构，专门用来描述加工逻辑。所以，它既有自然语言灵活性强、表达丰富的特点，又有结构化程序的清晰易读和逻辑严密的特点。其语法结构包括内层语法和外层语法。内层语法可以使用数据词典中定义过的词汇、易于理解的一些名词、运算符和关系符，它比较灵活；外层语法设定一组符号用于描述各种控制结构，具有较固定的格式。外层语法通常采用几种标准结构，如顺序结构、选择结构、循环结构，这些结构将加工中的各个操作连接起来。以下是一些典型的结构样式。

（1）选择结构，有以下 3 种类型。

```
如果  <条件>
      <操作>
```

① 肖汉．软件工程理论与实践．北京：科学出版社，2006

```
如果  <条件>
    则  <操作 A>
    否则<操作 B>
```

```
根据下列情况进行选择
条件 1<条件 1>
    <操作 1>
条件 2<条件 2>
    <操作 2>
……
条件 n<条件 n>
    <操作 n>
```

(2) 循环结构,有以下 2 种类型。

```
对每……
      <操作>
```

```
重复以下
      <操作>
直至<条件>
```

图 7-19、图 7-20 给出了自动柜员机的一些基本加工的结构化语言加工说明。

```
加工名:   验证密码
编号:     1.2
加工逻辑:
   输入密码
   如果密码正确
        则进入操作界面
        否则再次输入密码(如果密码输入的错误次数超过 3 次,则退卡)
```

图 7-19　ATM 基本加工 1.2 的加工说明

在进行描述加工说明时,需要注意以下几点。

(1) 一般来说,数据流图中的每一个基本加工都要配备一个加工说明。

(2) 加工说明应当能够表现出这个加工需要获得的输入数据流和产生的输出数据流。

(3) 加工说明应该主要描述加工应该做什么,而不是如何去做,在说明中不要涉及使用什么样的技术来实现,而是重点描述加工对数据流的处理策略。

(4) 加工说明应当简单、扼要、精练和具有较高的可读性。

```
加工名：    判断服务类型
编号：      3.2
加工逻辑：
   进入选择操作界面(共有 6 种基本操作)
                 1.取款
                 2.转账
                 3.余额查询
                 4.明细查询
                 5.修改密码
                 6.退卡
   如果需要进行以上某个操作
       则选择需要的操作
           选择 1.取款
                 进入取款处理加工
           选择 2.转账
                 进入转账处理加工
           选择 3.余额查询
                 进入余额查询处理加工
           选择 4.明细查询
                 进入明细查询处理加工
           选择 5.修改密码
                 进入修改密码加工
           选择 6.退卡
                 退卡
       否则 超过 10 秒钟没有操作或选择退出
                 退卡
```

图 7-20　ATM 基本加工 3.2 的加工说明

7.3.2 状态图

状态图反映系统因为外部的输入而由一个状态转换到另一个状态，其使用状态、事件等图形符号描述系统的行为。状态图通过系统的内部状态、外部事件来表述系统的工作流程。

在面向对象建模中，状态图可以用来对对象的状态和状态变化进行描述，为了不重复说明，状态图的具体内容详见面向对象分析方法部分。

7.3.3 实体-关系图(E-R 图)

数据流图明确各职能域间以及职能域内部数据流关系，它是业务需求和数据需求分析的关联，绘制数据流图是需求分析阶段的重要工作。而 E-R 图则是识别功能模型与数据模型间关联关系的，它可以更形象、更直接、更明确地表现需求分析人员的意图，同时也是需求分析人员对自己理解客户需求的再检查。E-R 图主要是由实体、属性和联系 3 个要素构成的。

1. 实体

实体是对问题域中具有一系列不同性质或属性的事物的数据抽象。例如，教务管理系统中的注册记录、学生、课程这 3 个现实对象，可以被看成是实体。

2. 联系

联系是指实体之间存在的联系，联系分为以下 3 种。

(1) 一对一联系(1∶1)。例如，一个公司只有一个总经理，公司与总经理的联系是一对一的。

(2) 一对多联系(1∶n)。在教务管理系统的例子中，学生实体与注册记录实体之间的联系就是一对多的，即一个学生可以有多个注册记录，而一个注册记录只能与一个学生对应。

(3) 多对多联系(m∶n)。例如，学生与课程之间的联系是多对多的，即一个学生可以上多门课程，多个学生也可以上一门课程。

3. 属性

属性是在实体、联系上所具有的一些特征值。例如，在教务管理系统中，记录号、注册时间是注册记录实体的属性；学号、姓名、性别、出生日期、入学时间、班级、系别是学生实体的属性；课程号、课程名、开课学期、学分、任课老师是课程实体的属性；注册记录号、学号是注册联系的属性；课程号、学号、选课学期是选课联系的属性。属性定义了实体、联系的性质，在设计属性时应该根据对要解决的问题的理解，来确定实体、联系的一组适当的属性。

在实体-关系图中，矩形表示实体，菱形表示联系，而属性用椭圆形来表示。如果某个属性是关键属性，则在该属性名称下用下划线来表示。图 7-21 是教务管理系统的部分实体-关系图。

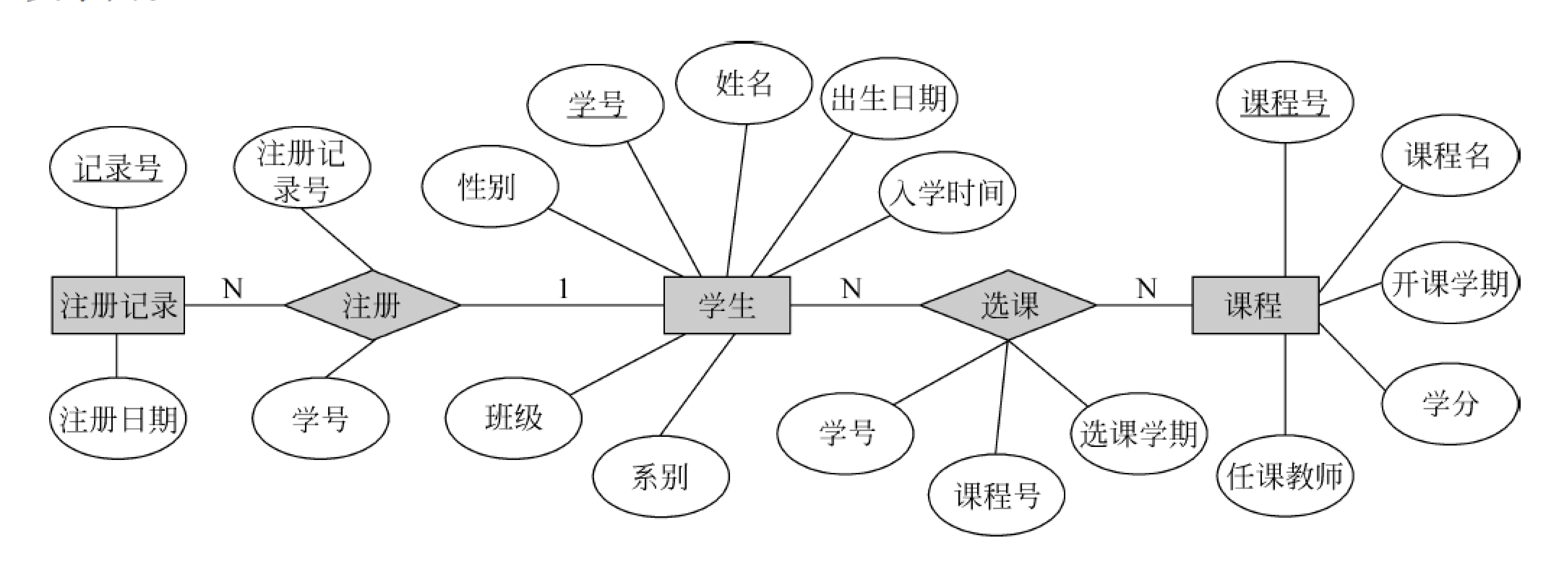


图 7-21　教务管理系统的部分实体-关系图

如果实体联系比较复杂，在绘制实体-关系图时可以只绘制出实体、联系及它们的关键属性，这样做的好处在于画面清晰，方便分析。例如，图 7-21 的简图就是图 7-22。

利用系统分析阶段建立的数据字典，并对照数据流程图对系统中的各个数据项进行分类、组织，确定系统中的实体、实体的属性、标识实体的码以及实体之间联系的类型。

在数据字典中“数据项”是基本数据单位，一般可以作为实体的属性。“数据结构”、“数据存储”和“数据流”条目都可以作为实体，因

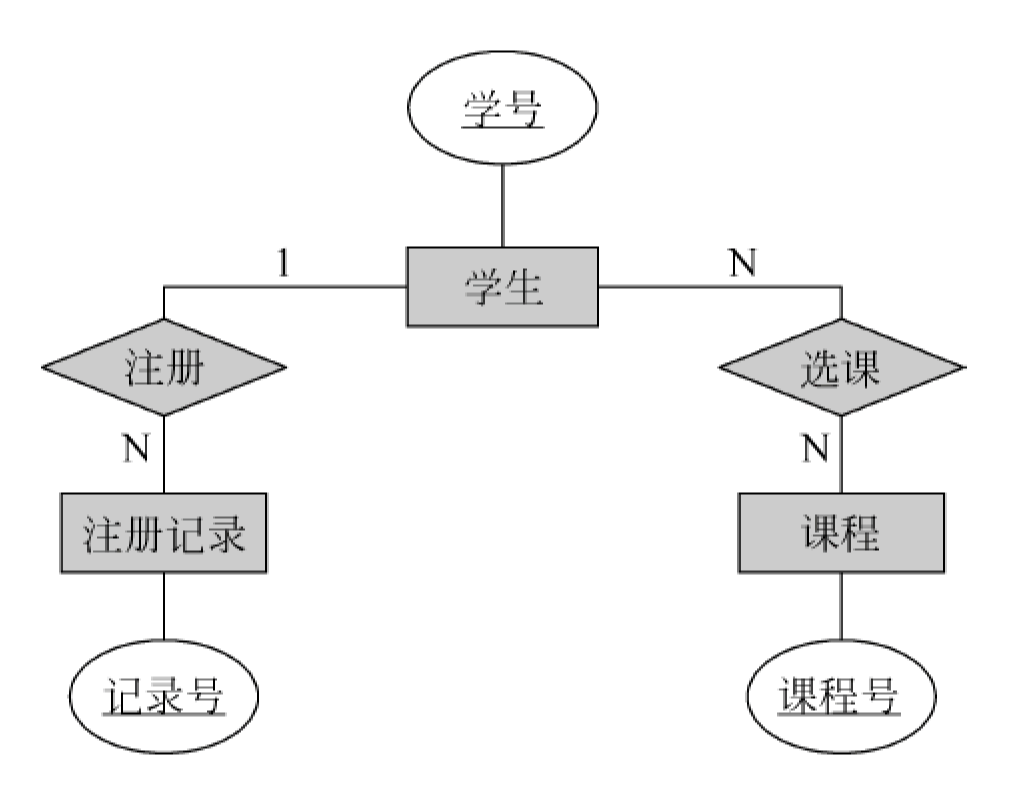


图 7-22　教务管理系统实体-关系简图

为它们总是包含了若干的数据项。作为属性必须是不可再分的数据项，也就是说在属性中不能包含其他的属性。

本章小结

系统论是以所研究和处理的对象作为一个系统，分析系统的结构和功能，研究系统、要素、环境3者的相互关系和变动的规律性，并优化系统观点看问题。系统工程的目的是解决总体优化问题，从复杂问题的总体入手，认为总体大于各部分之和，各部分虽然较劣但总体可以优化。系统工程层次结构：开发软件系统，需要有全局的眼光，检查整个业务和产品领域，保证能在“全局视图”上理解产品所在的位置；然后重点分析所关心的具体领域，在这个确定的领域上分析出所需系统的要素（如数据、软件、硬件、人员等）；最后对所需系统进行需求分析、系统设计和构建。

需求分析指的是在建立一个新的或改变一个现存的计算机系统时描写新系统的目的、范围、定义和功能时所要做的所有的工作。需求分析是软件工程中的一个关键过程。在这个过程中，系统分析员和软件工程师确定顾客的需要。

需求分析阶段的工作，可以分成以下4个方面：问题识别、分析与综合、编制需求分析阶段的文档、需求分析评审。

结构化分析方法：该方法的要点是面对数据流的分解和抽象；把复杂问题自顶向下逐层分解，经过一系列分解和抽象，到底层的就都是很容易描述并实现的问题了。结构化分析使用数据流图、数据字典、实体-关系(E-R)图、状态图等工具，来建立一种称为结构化说明书的目标文档——需求规格说明书。

数据流图(DFD图)是用于表示软件系统逻辑模型的一种图形，一个基于计算机的信息处理系统由数据流和一系列的转换构成，这些转换将输入数据流变换为输出数据流。

(1) 画数据流图的思考顺序：首先应画出系统的输入数据流和输出数据流，然后再考虑系统的内部；每一个加工也是先画其输入输出，再考虑其内部。

(2) 数据流图的画法：可以考虑使用“自顶向下逐层分解”的结构化分析方法。这种方法将大问题分割成中问题，中问题分割成小问题，这体现了分解和抽象的精神。具体来说，就是将数据流图分层描绘，一般由顶层、底层、中间层组成。

描述加工规约的工具主要有结构化语言、判定表、判定树。

状态图反映系统因为外部的输入而由一个状态转换到另一个状态，其使用状态、事件等图形符号描述系统的行为。

实体-关系图是识别功能模型与数据模型间的关联关系，主要是由实体、属性和联系3个要素构成的：①实体是对问题域中具有一系列不同性质或属性的事物的数据抽象；②联系是指实体之间存在的联系；③属性是在实体、联系上所具有的一些特征值。

思考与练习

1. 某服装制造企业的组织机构包括经理室、办公室、生产一部、生产二部、生产三部、营销部、管理部、财务部。该单位首先针对系统总体目标，进行了网络建设规划。网络总体设

想：为各分公司建立局域网。总公司以管理部为广域网主体，实现对各类管理、公司权限级的信息（包括行政信息、办公自动化信息、管理信息）、共享数据的管理。分公司建立局域网，对上述信息以外的信息进行汇集，并根据各分公司的管理对象、特点和要求，各自扩展网络管理的内容和信息的汇集、处理。本次项目目标是开发一个办公自动化系统，该系统通过以电子邮件机制为基础的办公自动化技术的应用，从交流（Communication）、协调（Coordination）、控制（Control）入手，实现公司内部的电子信息交换和现代化办公管理，提高系统管理效率。检查该服装制造企业的业务和产品领域，保证能在"全局视图"上理解项目所在的位置。重点分析所关心的具体领域，在这个确定的领域上分析出系统所需的要素。

2. 选择一个大型系统或产品，定义描述系统或产品的整体视图的一组领域，描述构成一个或两个领域的元素集合，对一个元素，标识必须开发的技术构件。

3. 有哪些常用的需求收集方法和技术？试选择某一系统并根据方法进行需求收集。

4. 在软件需求分析过程中最强调信息的沟通，但为什么通信路径经常中断？

5. 在进行需求分析时，往往会遇到一些行政问题的阻碍，如员工对新信息系统应用的积极性不高甚至抵制，导致需求分析无法继续。什么原因导致了这样的问题？怎样使这样的行政问题的危害减少到最小程度？

6. 一个机票预订系统完成如下功能：工作人员把预订机票的旅客信息（姓名、身份证号码、航班号、出行时间、出行起始地、出行目的地等）输入该系统，系统为旅客预订航班（如果航班满员，系统自动查询出满足出行时间、出行起始地、出行目的地的其他航班供选择），打印出取票通知，旅客在飞机起飞前一天凭取票通知缴款取票，工作人员通过使用该系统核对旅客信息无误后，确认缴款并打印出机票给旅客；系统需要工作人员登录才能进入，系统登录后或完成上一次操作后，都会返回到选择操作界面（选择"预订航班"还是"缴款"）。请用数据流图描绘该系统。

7. 某储蓄所的存、取款业务需求如下：储户将填好的存（取）款单及存折交给业务员，业务员进行分类处理。如存（取）款单填写有误，将存折及存（取）款单返还储户；如果是存款，将存折及存款单、现金交存款业务员处理，存款业务员通过读存折器查出储户账款信息，录入存款信息，打印存折后将存折还给储户；如果是取款，将存折及取款单交取款业务员处理，取款业务员通过读存折器查出储户账款信息，录入取款信息，打印存折后将存折与现金交付储户。试画出该系统分层的数据流图。

8. 某学校的教材购销系统有如下功能：学生买书，首先填写购书单，计算机根据各班学生用书表及售书登记表审查有效性。若有效，计算机根据教材库存表进一步判断书库中是否有书；若有书，把领书单返回给学生，学生凭领书单到书库领书。对脱销的教材，系统用缺书单的形式通知书库，新书购进库后，也由书库将通知返回系统。就以上系统功能画出分层的数据流图。

第8章 面向过程的结构化设计

智慧就是懂得该忽略什么的技巧。

——William James①

8.1 软件设计的基本概念和原理

在完成了需求分析之后，项目开发人员对系统的需求有了完整准确的理解，即知道了"做什么"的问题，接下来就是回答"怎么做"的问题。对于一个较大的软件项目来说，软件设计一般被分为两个阶段进行。第一个阶段为总体设计阶段，期间项目开发人员确定软件系统的基本框架；第二个阶段为详细设计阶段，期间确定软件系统的内部实现细节。无论采用何种具体的软件设计方法，抽象与求精、信息隐蔽、模块化设计、模块独立性等有关原理都是设计的基础，为"程序正确性"提供了必要的框架。

8.1.1 抽象

抽象是人在认识复杂世界时所使用的最有力的工具。抽象是从众多的事物中抽取出共同的、本质性的特征，而舍弃其非本质的特征。例如苹果、香蕉、梨、葡萄、桃子等，它们共同的特性就是水果。得出水果概念的过程，就是一个抽象的过程。要抽象，就必须进行比较，没有比较就无法找到共同的部分。

共同特征是指那些能把一类事物与他类事物区分开的特征，这些具有区分作用的特征又称为本质特征。因此抽取事物的共同特征就是抽取事物的本质特征，舍弃不同特征。所以抽象的过程也是一个裁剪的过程，不同的、非本质性的特征全部被裁剪掉了。

共同特征是相对的，是指从某一个侧面看是共同的。例如，对于汽车和大米，从买卖的角度看都是商品，都有价格，这是它们的共同的特征，而从其他方面来比较时，它们则是不同的。所以在抽象时，同与不同，决定于从什么角度上来抽象。抽象的角度取决于分析问题的目的。

我们要给出一个问题的解决方案时，可以有很多层次的抽象级。在较高的抽象级上以概括性的、适合问题所处环境的语言来描述解决方案；较低的抽象级上提供较详细的解决

① William James(1842—1910)，美国本土第一位哲学家和心理学家，也是教育学家，实用主义的倡导者。

方案说明。在随后讲到的模块化概念是抽象概念的一种表现形式。在软件开发的过程中，采用自顶向下、由抽象到具体的方式来进行思考构建系统，对软件的设计和实现起到了优化的作用，因为在不同的抽象层次上有不同的解决方案，这就提高了软件的可理解性，也使软件更容易维护。

8.1.2　信息隐蔽

信息隐蔽是指每个模块的内部实现细节对外部来说是看不见的，即模块内部的数据、代码等信息不允许其他不需要这些信息的模块使用。这样主要有两个好处，一是利于模块之间相互有效隔离，使每个模块更加具有独立性，因为模块之间的通信更简练，模块与模块只传递必需的信息；二是可以使系统具有更好的健壮性，以及更好的可维护性，原因是如果某个模块出现了错误，因为信息隐蔽的作用，不会使被隐蔽的错误信息在整个系统中扩散，从而导致整个系统的崩溃，同时根据出错信息可以快速地找出错误的模块，这样软件纠错工作的效果、质量都会得到提高。另外，用户的需求可能会随时间变化而变化，那么这时一些模块的功能就需要做出适时的调整和改造。信息隐蔽会使对模块改造所带来的影响限制在需要改造的模块内，使软件系统的升级更加便利。

8.1.3　模块化设计

人们不断创新的目的在于更加高效。从软件开发的角度来看，人们希望开发维护同样一个系统，所使用的时间最短，所耗费的成本最低。软件系统模块化就是出于这样的目的，提出了一种提高开发效率的思想。

在结构化分析方法中，模块的规模可大可小，是一个功能单位。模块可以是软件系统中的一个子系统，也可以是子系统内一个功能程序块（由边界元素限定的数据说明、可执行的语句等的序列，而且有一个总体标识符来代表它。像 Pascal 语言中的 Begin…End 对，或者 C，C＃和 Java 语言中的{…}对），并且可以是功能程序块内的一个程序基本单元（如函数、过程）。我们把模块看成是构成程序的基本构件。把软件系统划分成独立命名、访问的模块就是“模块化”。每几个子功能用一个模块来实现，当将所有模块整合在一起时，就构成了软件系统的所有功能。模块化使复杂的大型软件系统能被高效地开发和管理。如果一个复杂的大型软件系统没有被模块化，或者换个角度说，它只有一个模块，那么这样程序将很难被正确理解、分工开发、高效维护。模块化实际上体现出了系统所具有的功能层次结构。模块化体现了人类解决复杂问题的方法，也就是将大的复杂的软件问题分解成许多小的简单的软件问题。

模块化可以简化软件问题。那么是否可以依靠模块化使系统不断分解而使整个系统不断简化？但是有一个问题，即我们所要开发的复杂系统往往是一个整体，它往往不是简单的模块累加，模块与模块之间往往存在联系，这种联系又往往表现为通信，一个系统模块越多，模块与模块之间的通信或接口就越多，希望依靠模块化将系统不断被分解而使软件成本不断降低的愿望可能是不切实际的。随着系统的分解，系统中模块数目将会增加，模块接口也会增加，软件构造会由此变得复杂起来，模块连接的难度也会由此加大。

图 8-1 是模块与成本的关系图，图中有 3 条曲线：软件总成本、接口成本、每个模块的成本。其中：软件总成本＝接口成本＋（模块数×每个模块的成本）。若系统增加模块的数目，则每个模块需要的成本的确减少了，但模块间接口所需的成本也增加了。从图中可以看出，要使软件总成本最低，并非一味增加模块的数目，而是要确定一个最合理的模块数目，才能使系统的开发成本最小。那么如何确定一个软件系统的模块数目？目前还不能精确地确定出，但有些标准或原则可以用来帮助我们评价系统的模块划分合理性。这里定义了 5 条这样的标准，分别从模块可分解性、可组装性、可理解性、连续性、保护性来理解。

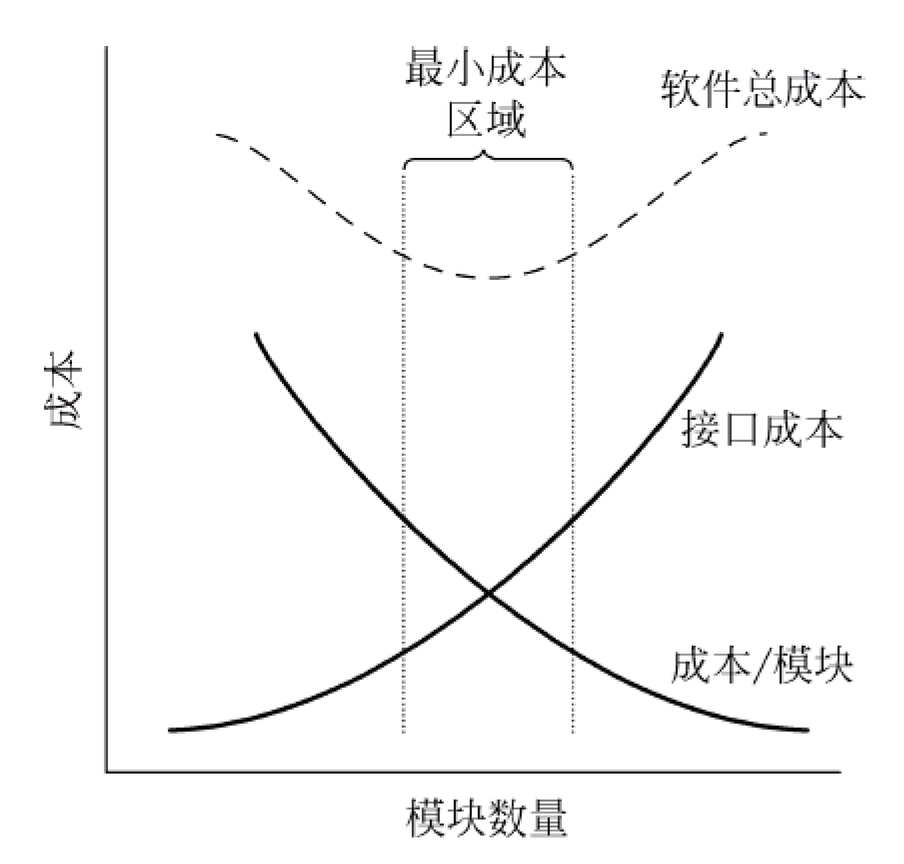


图 8-1　模块与成本的关系图

(1) 如果一种设计方法提供了把问题分解为子问题的系统化机制，它就能降低整个问题的复杂性，从而可以实现一种有效的模块化解决方案。

(2) 如果一种设计方法能把现有的（可重用的）设计构件组装成新系统，它就能提供一种并非一切从头开始的模块化解决方案。

(3) 如果可以把一个模块作为一种独立单元（无须参考其他模块）来理解，那么，这样的模块是易于构造和易于修改的。

(4) 如果对系统需求的微小修改只导致对个别模块，而不是对整个系统的修改，则修改所引起的副作用将最小。

(5) 如果在一个模块内出现异常情况时，它的影响仅局限在该模块内部，则由错误引起的副作用最小。

8.1.4　模块独立

模块独立是模块化、抽象、信息隐蔽的直接结果，是指系统中的模块尽可能地只涉及自己特定的子功能，并且模块接口简单，与其他模块没有过多的通信。如果系统中每个模块都具有很好的独立性时，系统实现起来就更加容易。因此，模块独立性是衡量软件中模块质量最重要的指标。

一般来说，采用耦合和内聚这两个定性的技术指标来对这一模块的独立性进行衡量。内聚性可以衡量各模块内部功能的结合强度，模块内部各元素之间结合得越紧密，则它的内聚性就越高；耦合性显示了模块与模块之间相互的依赖关系，模块与模块之间联系越紧密，耦合性就越高。所以，为了使模块具有更强的模块独立性，就要求模块有较高的内聚性和较低的耦合性。

8.1.5　耦合

耦合度量了各模块之间相互关联的程度，各个模块之间接口的复杂程度、接口数据对模块内部运算的影响程度、使用模块的方式都决定了耦合的强弱。以下是耦合的几种主要形式。

(1) 非直接耦合。两个模块之间的联系，仅限于被共同模块控制和调用，它们之间没有直接的联系，那么这种耦合就成为非直接耦合，因为模块与和模块之间没有数据通信，所以它的耦合形式是最弱的。

(2) 数据耦合。模块与模块之间发生联系,彼此之间通过接口参数实现通信,传递的接口参数数据是用于计算的,它们不会影响内部程序执行的路径。我们提倡使用数据耦合,因为它是一种较弱的耦合。

(3) 控制耦合。如果在数据耦合的基础上,模块间接口参数不仅传递数据,同时还传递标志、名字、开关等控制信息,从而影响模块的内部程序执行路径。显而易见,控制耦合比数据耦合的耦合性要强一些,它属于中等程度的耦合。如果需要通过接口传递模块内多项功能的选择时,就需要用到控制耦合。

(4) 公共耦合。在软件系统中,可能有独立于模块而存在的数据文件、公共变量公共数据环境。模块之间通过访问公共数据环境从而实现通信。使用公共耦合的原因在于,模块之间通过接口传递参数不方便,模块之间需要公共数据环境来共享数据。因此,相对于接口耦合来说,公共耦合使模块的独立性下降,它也属于中等程度的耦合。

(5) 内容耦合。内容耦合是一种耦合性很强的耦合,这种耦合严重影响了模块的独立性。它的表现形式主要有以下几种:①模块直接访问另一个模块的内部数据;②模块不通过正常的入口转到另一模块内部;③模块之间存在一部分代码重叠;④某个模块有多个入口。内容耦合致使模块的变动变得非常困难,程序维护和升级也极其困难,这要求我们在设计软件结构时,不允许出现内容耦合。

8.1.6 内聚

内聚性是信息隐蔽概念的自然扩展,它度量了模块内部各个元素彼此结合的紧密程度,元素之间联系越紧密,其内聚性越强。因此,我们需要尽可能地设计内聚性强的模块,从而达到模块独立,功能集中的目的。那么如何才能够提高模块的内聚性呢?这依赖于设计人员对模块功能有正确的、深入的认识,为了更好地认识模块的内聚性,我们把模块内聚分为以下几种主要类型。

功能内聚、信息内聚、通信内聚、过程内聚、时间内聚、逻辑内聚和偶然内聚。从内聚性的高低来说,以上几种类型的内聚性,根据从前到后的顺序,其内聚性是由高到低的,而其模块独立性则是由弱到强的。可以这么说,模块的内聚程度越高,独立性就越强。下面对上述几种内聚类型分别加以说明。

(1) 偶然内聚。偶然内聚即模块内部各元素之间的联系很少或者没有。偶然内聚产生的原因是程序员对程序没有精心地进行软件结构的设计,或者设计软件结构是很随意的,模块的功能定义模糊、内聚程度很低。

(2) 逻辑内聚。逻辑内聚是将几种相关的功能组合在一起形成一个模块。一般来讲,模块内部功能之间是选择关系,在调用时需要对模块传递参数来确定所选功能。由于模块内部各功能执行的时间是相关的,因此逻辑内聚比偶然内聚的内聚性要高,但在调用该模块时,需要传递控制参数,所以增强了模块间的耦合强度。

(3) 时间内聚。时间内聚是指模块内部各功能之间的执行与时间相关。初始化模块就是时间内聚的典型例子,它的功能包括给变量和初值、打开通信连接、初始化对象、打开文件等,所有这些功能都需要在系统启动时完成,时间内聚比逻辑内聚具有更强的内聚性,但它也属于低内聚类型。

(4) 过程内聚。如果模块内的各个元素的执行是按照一定次序来进行的,即各个元素

的处理是相关的，则称其为过程内聚，例如，用流程图来描绘系统的某些功能，流程图中的循环部分、分支部分、计算部分可以看成 3 个过程内聚模块。过程内聚比时间内聚的内聚性更强一些，但是从模块间的耦合性来看，每一个过程内聚模块仅仅完成某一功能的一小部分，因此模块与模块之间的耦合程度还是比较高的。

(5) 通信内聚。一个模块内部可能有几个功能部分，如果这些功能部分都使用相同的数据输入，或者产生相同的数据输出，这就是通信内聚。它的内聚程度比过程内聚要高。

(6) 顺序内聚。如果模块内某一功能元素的输出作为另一个功能元素的输入，模块内各功能元素顺序联结，它们之间关系紧密，那么称这个模块为顺序内聚模块。

(7) 功能内聚。如果为了实现模块的具体功能，模块内各个元素都是必需的，这些元素要协同工作，它们无法单独执行，称这样的模块为功能内聚模块。功能内聚模块的内聚程度很高，在进行软件设计时，应尽可能地实现功能内聚。

8.1.7 结构图

结构图是描绘软件体系结构的一种工具，它用图形的表示方法来描绘软件的体系结构，它描绘出系统的组织结构和各元素之间的相互关系，图 8-2 是结构图的一个例子。

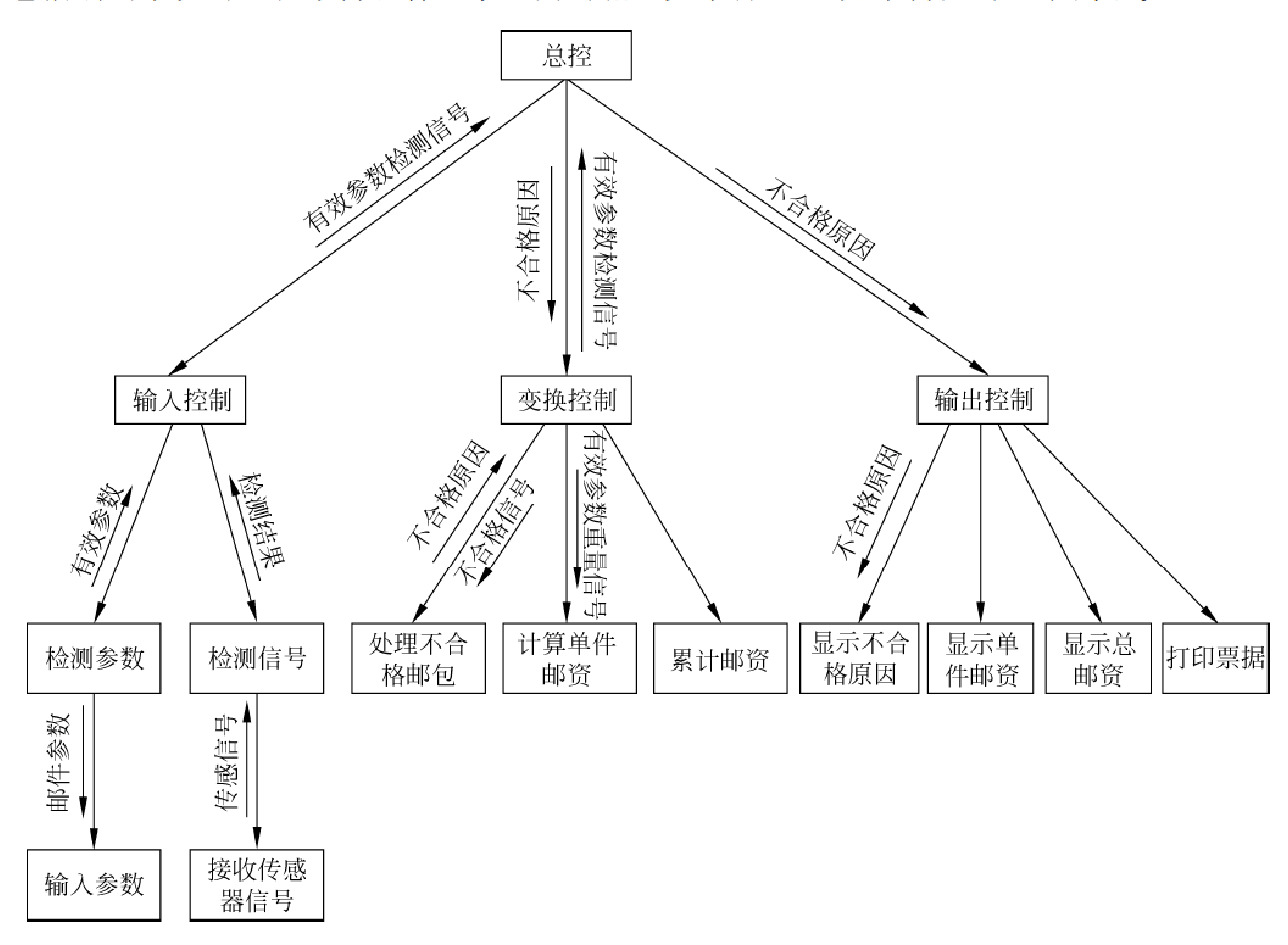


图 8-2 邮件检测与计价电子仪器系统软件体系结构

以下是结构图的主要成分。

(1) 模块。模块使用方框来标识，方框中标明了模块的名称。它反映了模块的功能。

(2) 调用。用一个模块指向另外各模块的箭头来表示，其中，前一个模块是调用模块，

后一个模块是被调用模块。通过这种箭头表示,可以知道模块之间的调用关系,但具体的被调用次数却不知道。

(3)通信。通信用小箭头来表示,它一般绘制在调用箭头的边上,通信小箭头指明了从一个模块传送给另一个模块的数据或标志以及传送的方向。

(4)判断调用。用菱形来表示。根据调用模块内部的判断条件,如果判断条件成立,则调用某一从属模块,否则不调用。

(5)循环调用。用弧形箭头来表示。如果一个模块需要循环调用若干个从属模块,这就需要使用循环调用来表示。

在绘制结构图时应该注意以下几个要点。

(1)在绘制模块的左右位置时,可以依据传递数据的先后顺序来确定,越先被调用的模块,出现在最左边,越后被调用的模块,出现在最右边。

(2)同一名字的模块在图中只能出现一次。

(3)为了方便使用者阅读,结构图尽可能画在同一张纸上。

8.2　软件总体设计的任务和目标

在总体设计阶段中应从系统开发的角度出发,将系统逐次分割成层次结构,系统被表达为一个结构清晰层次分明的模块组合,每个模块完成各自相对简单的功能,并且它们之间都保持一定的联系,另外还定义这个系统与外部系统的接口。它的工作包括以下具体内容:确定模块的层次结构、每个模块的功能、模块间的调用关系、模块间的接口,另外还需要明确系统所需的算法,需要明确系统所需模块间的控制方式,明确外部信息的接收和发送的方式,明确输入输出数据的详细数据结构,以及它的使用规则。明确了以上内容后,就要编写系统的设计阶段文档,它们包括系统设计说明书、数据库设计说明书、初步的测试计划等。最后组织共利益者(“共利益者”指的是受到某种负责产生输出的方式影响的群体或个人,共利益者可能包括项目经理、供方、顾客以及其他人)对软件设计的过程文档进行评审。

这一阶段要从需求分析转化到设计模型,在这里主要介绍从结构化分析转化到结构化设计的方法。图 8-3 是分析模型元素和软件设计内容的示意。表 8-1 是从分析模型转化到设计模型的元素对应关系。

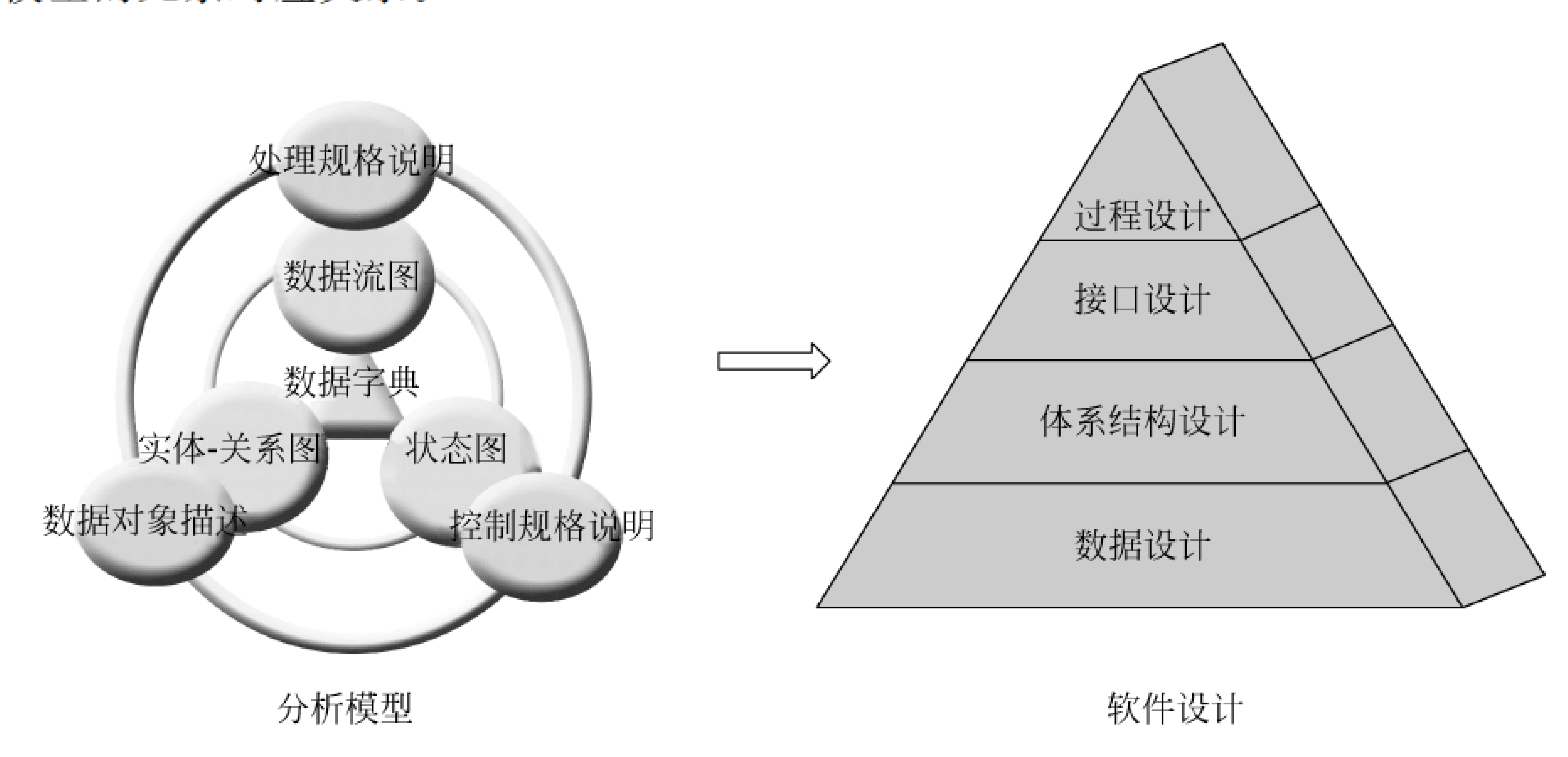


图 8-3　分析模型元素和软件设计内容的示意

表 8-1 从分析模型转化到设计模型的元素对应关系

分析模型元素	转化成设计模型的元素
状态转换图、控制规格说明、处理规格说明	过程设计
数据流图	接口设计
数据流图	体系结构设计
实体-关系图、数据对象描述、数据字典	数据设计

软件系统的设计必须根据需求分析的结果来进行，在第 7 章中给出了需求分析模型的表述，图 8-3 描绘了从结构化分析转化到结构化设计过程中的元素对应关系，如图所示，数据字典、实体-关系图、数据对象描述转变成设计阶段的数据结构，这就是在进行数据设计；从数据流图分析归结出系统的体系结构，体系结构设计确定了系统模块之间的关系和接口，描述了系统内部各组成元素之间、软件与外部系统之间以及软件与使用者之间的通信方式；数据流图提供了接口设计所需要的信息；而过程设计是对软件构件的过程性说明，可以从处理规则说明、控制规格说明和状态图获得过程设计所需要的信息。根据以上的思路，就可以把分析阶段的数据模型、功能模型和行为模型传递给系统设计人员，要他们把各个模型所表达出来的信息，使用合适的方法完成数据设计、体系结构设计、接口设计和过程设计。

8.3 结构化软件设计

结构化软件设计，是面向数据流的设计方法，数据流图是设计的基础。结构化设计定义了不同的映射，利用这些映射可以把数据流图变换成软件体系结构。

根据数据变换的性质，可以把数据流图分为变换型和事务型两类，所以结构化设计也被分为变换设计、事务设计以及两者相结合的综合设计。

8.3.1 基本概念

面向数据流的设计方法，把信息流映射成软件体系结构，不同的信息流选用不同的映射方法，总体来说，有以下两种类型的数据流。

1. 变换流

变换流体现的是数据从输入到加工，再到输出的一般步骤，数据首先需要输入过程，由外部形式变换成内部形式，这种内部形式适合进行加工处理；然后经过变换中心，将输入的数据加工成一种新的数据形态；接着再通过输出通道变换成外部形式。当数据流图具备这些特性时，这种信息流就称为变换流，如图 8-4 所示。

2. 事务流

当输入的信息流可以引发多个不同的事务活动流程，并且数据流图中有一个事务调度中心，则称这种信息流为事务流，如图 8-5 所示。

不同的信息流对应的分析方法是不同的，需要根据数据流图中的流的类型分别进行变换分析或事务分析，从数据流图到最终的软件体系结构的分析设计过程如图 8-6 所示。

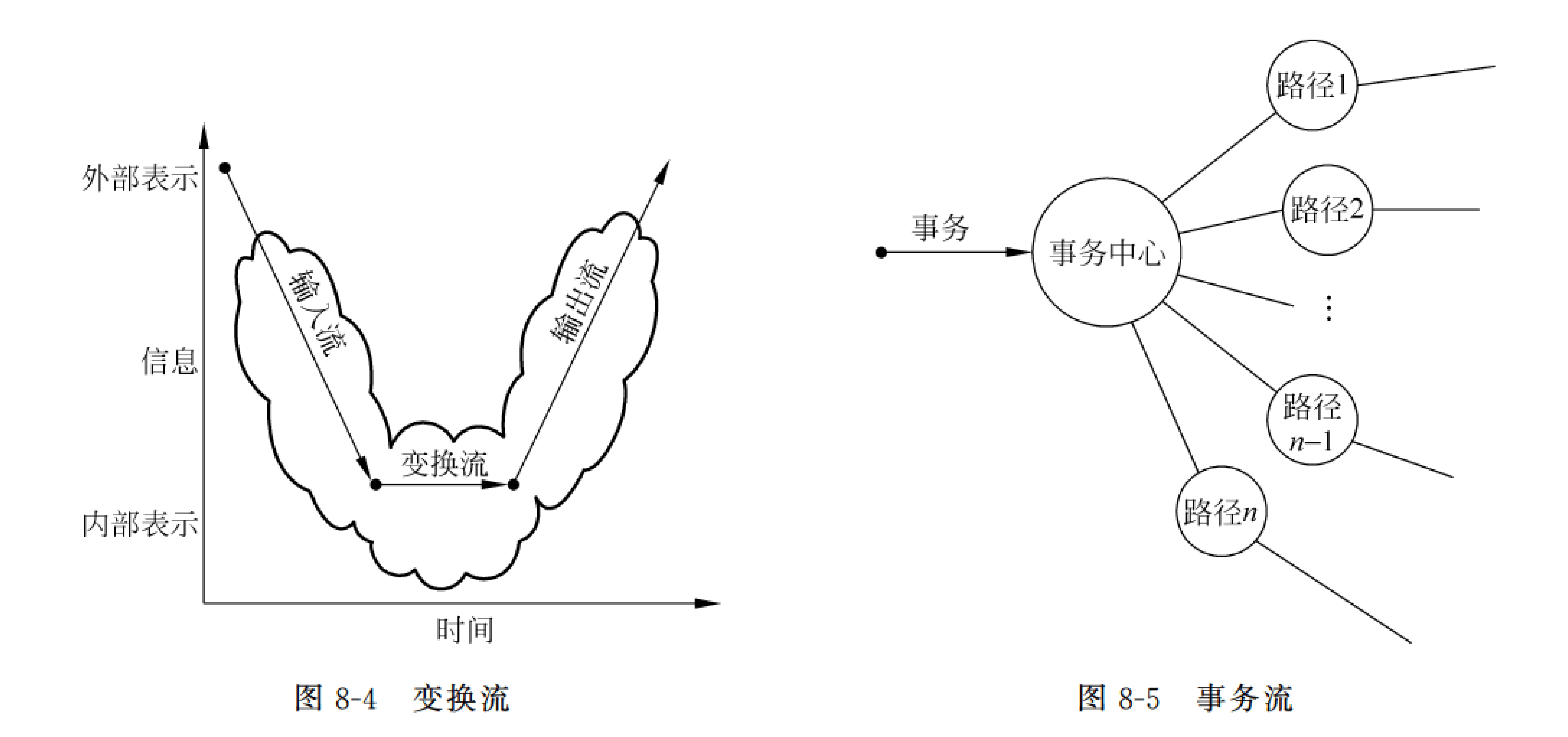


图 8-4　变换流　　　　图 8-5　事务流

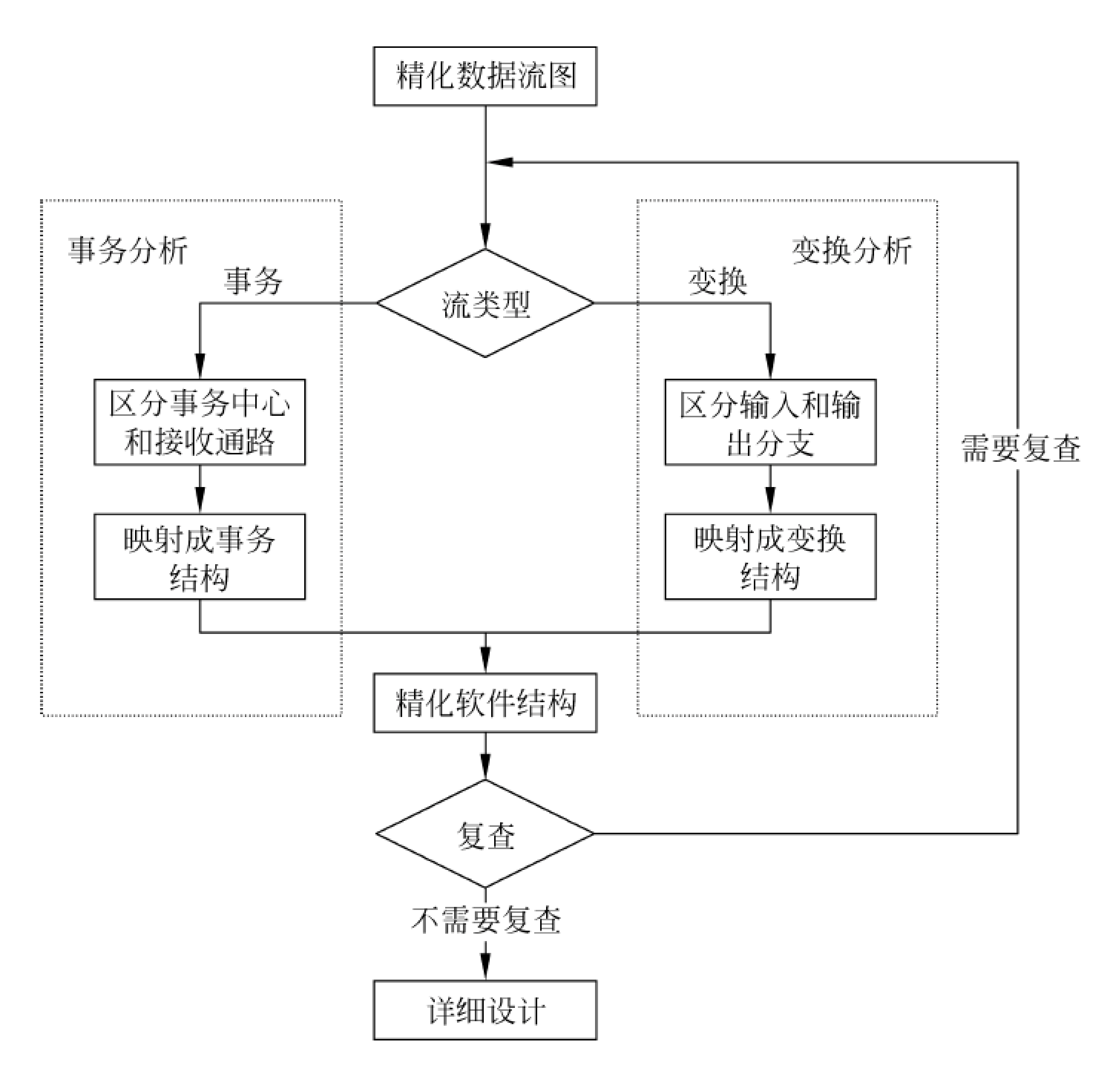


图 8-6　面向数据流方法的设计过程

以下通过 ATM 系统的例子，分别阐述变换分析和事务分析。首先看 ATM 系统的 1 层 DFD(图 8-7)。

从图中可以明显地看出，1 层 DFD 具有明显的变换流特性，包含了 3 部分内容：①以等待验证为主的数据流，对应了变换流中的输入流；②以服务处理、异常处理为主的数据流，对应了变换流中的变换中心；③以输出处理为主要的数据流，对应了变换流中的输出流。变换流的 3 部分在图中以虚线分隔。

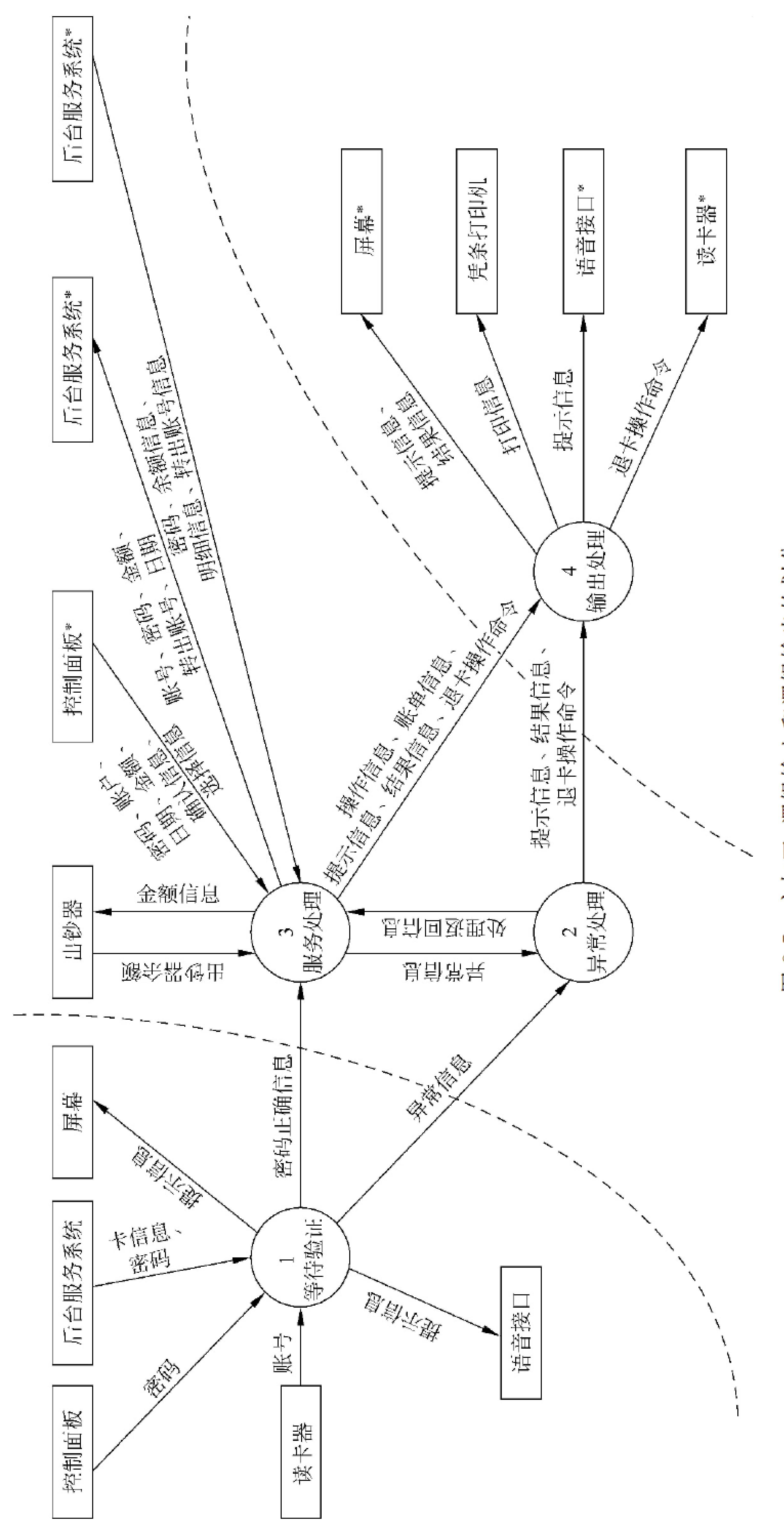


图 8-7 主加工、逻辑输入和逻辑输出的划分

8.3.2 变换分析

变换流具有明显的输入、加工(变换)、输出的界面,是一种线状的结构 I→P→O。下面介绍变换分析的步骤。

第一步:找出系统的变换中心,确定输入流和输出流。

一种方法是,根据系统的需求说明书可以很容易地先决定哪些加工是系统的变换中心,然后决定输入流和输出流,例如,多条数据流汇合处往往是系统的变换中心,流入变换中心的数据流就是输入流,从变换中心流出的数据流就是输出流。

另一种方法是,如果变换中心一时不能确定,则可以先确定输入流和输出流,这种方法的具体步骤如下。

(1) 确定输入流

从物理输入端开始,向系统的中间移动,直到某个数据流不能被看成是系统的输入,这部分的数据流就可以认为是输入流。如图 8-7 所示,图的左边部分可以确定为逻辑输入,它包括数据流"账号"、加工"1 等待验证"、数据流"密码正确信息"和"异常信息",其中加工"1 等待验证"被称为预加工或辅助加工。

(2) 确定输出流

从物理输出端开始,向系统的中间移动,直到某个数据流不能被看成是系统的输出,这部分的数据流就可以认为是输出流。如图 8-7 所示,图的右下部分可以确定为逻辑输出,它包括数据流"操作信息、账单信息、提示信息、结果信息、退卡操作命令"、"提示信息、结果信息、退卡操作命令"、加工"4 输出处理"、数据流"提示信息、结果信息"、"打印信息"、"提示信息"、"退卡操作命令",其中加工"4 输出处理"被称为辅助加工。

(3) 确定变换中心

确定了输入流和输出流后,中间的部分就是变换中心。如图 8-8 所示,图的中间部分可以确定为变换中心。

第二步:设计变换流模块的基本结构。

首先要确定一个顶层——主模块,这个主模块反映了该模块要做的工作,对于变换流性质的模块,其第一层的基本结构可以分 3 部分来设计,即:①为每个输入流设计一个输入模块,负责向主模块提供数据;②为每个输出流设计一个输出模块,负责从主模块输出数据;③为变换中心设计一个变换模块,负责将输入流变换成输出流。顶层主模块负责协调控制第一层 3 个模块的工作。在 ATM 的例子中,从变换流分析出来的软件结构如图 8-8 所示。

第三步:对每个模块进行进一步的分解。

根据"自顶向下逐步细化"的原则,细化需要细化的每个模块。在一开始是对输入模块、变换模块、输出模块进行细化。在 ATM 的例子中,我们分析系统的 1 层 DFD(图 8-9)后,认为"输入模块"、"变换模块"和"输出模块"需要进一步细化。通过评价和精化后的软件结构如图 8-10 所示。

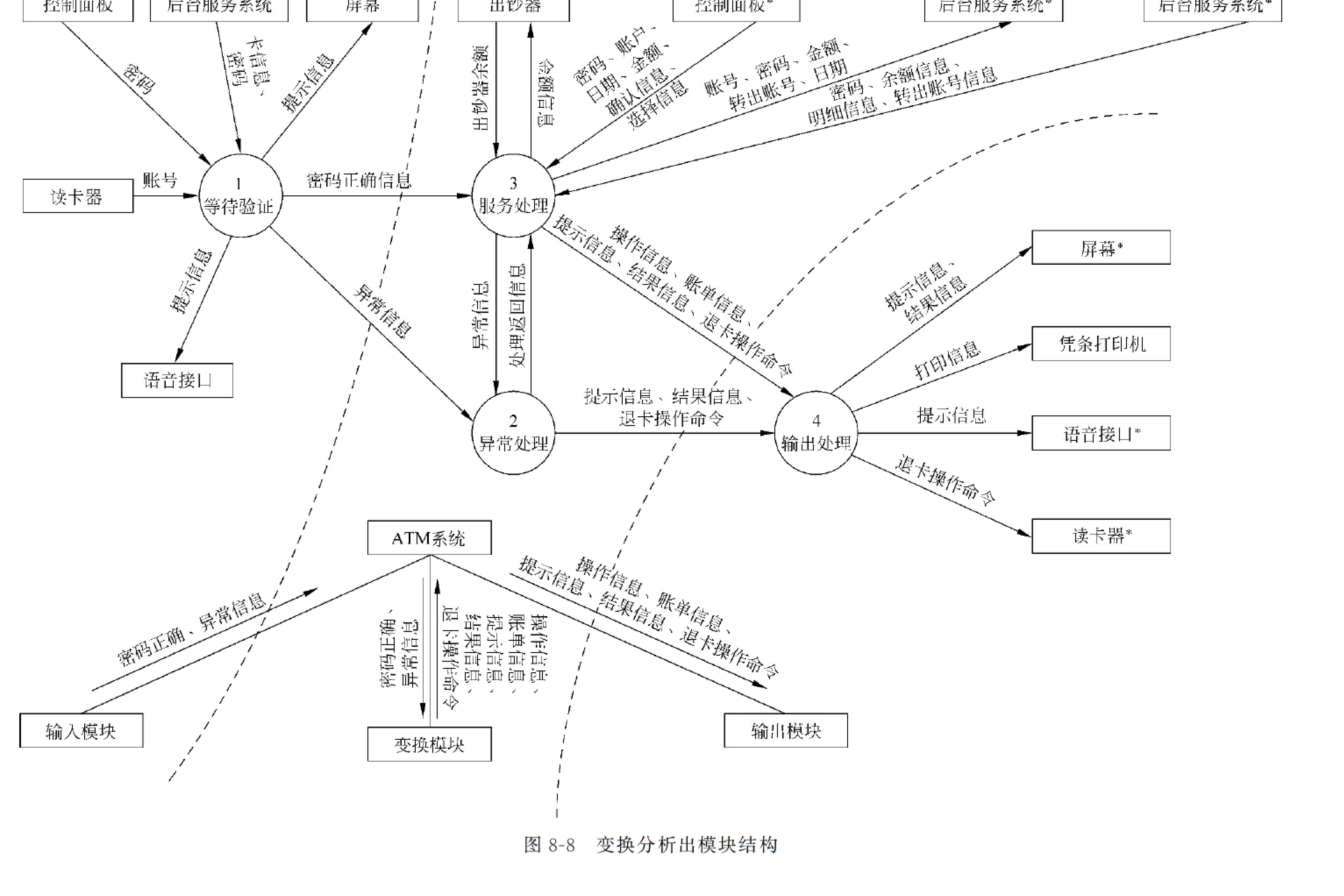


图 8-8 变换分析出模块结构

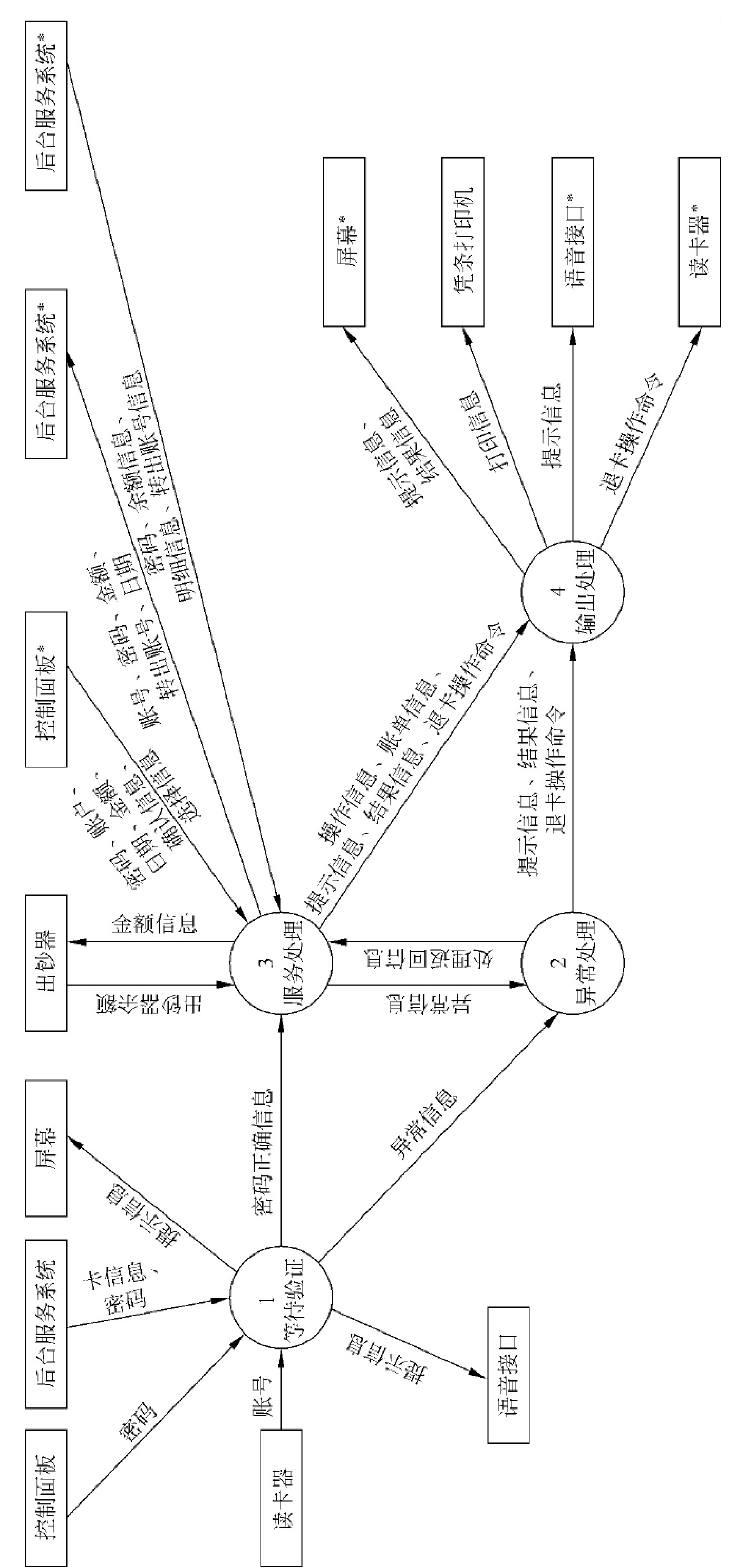


图 8-9 ATM 系统 1 层 DFD

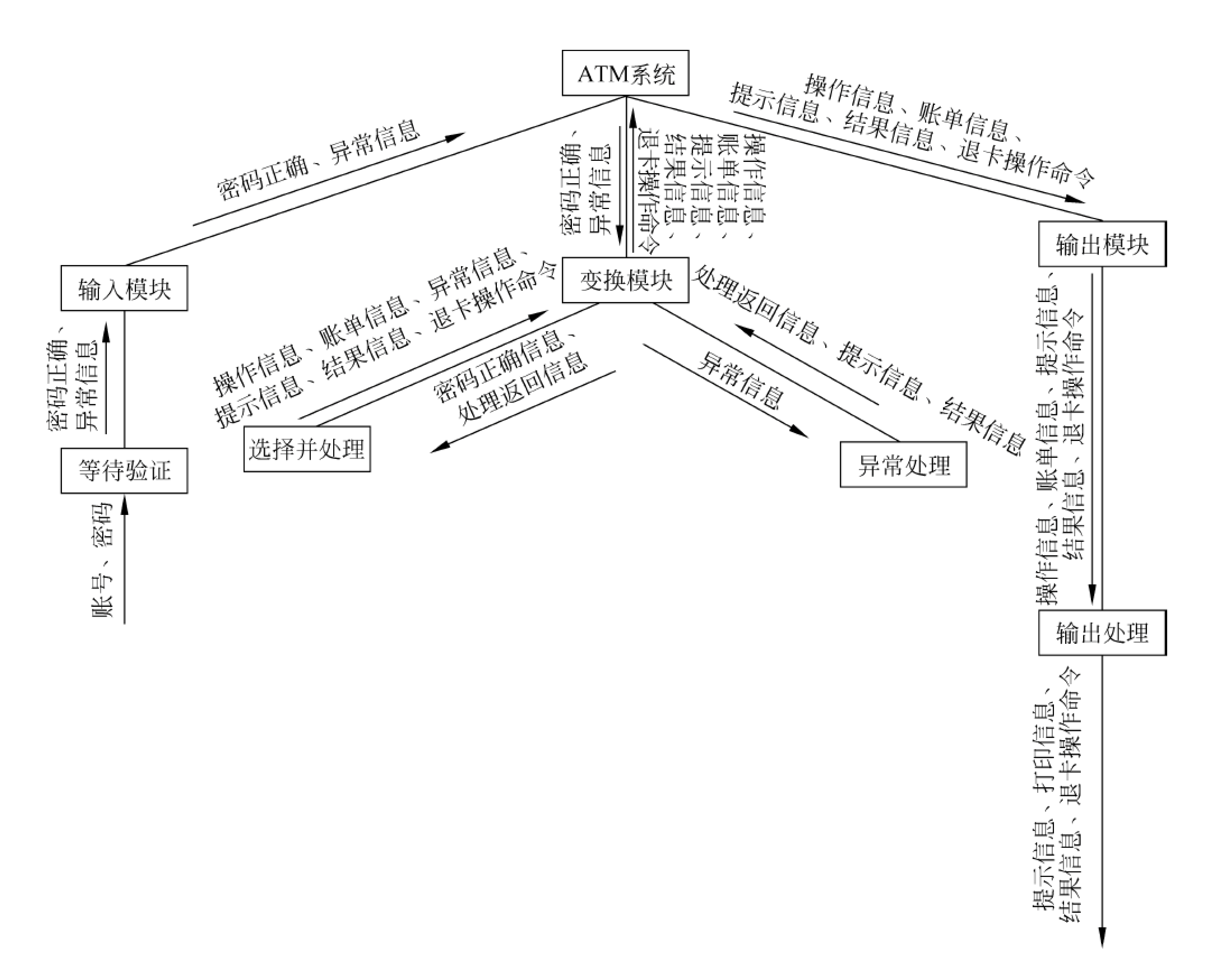


图 8-10　细化后的结构图

我们仔细分析了系统的 1 层 DFD 后，发现对应于“选择并处理”模块的那部分 DFD 具有明显的事务特征，应该使用事务分析来进行结构设计。

8.3.3 事务分析

虽然在所有情况下，都可以使用变换分析方法设计软件体系结构，但是如果数据流具有明显的事物特征时，也就是有一个明显的事务中心时，还是采用事务分析比较恰当。

事务分析的步骤和变换分析方法基本类似，在依据“自顶向下逐步细化”的原则下，事务分析主要分为以下 3 个步骤。

第一步：确定出事务中心和各活动路径。

在 ATM 系统的例子中，如图 8-11 所示，可以确定“判断服务类型”是事务中心，“取款处理”、“转账处理”、“余额查询处理”、“明细查询处理”、“修改密码”是各活动路径。

第二步：设计事务流模块的基本结构。

首先确定事务流模块的一个顶层——主模块，在如图 8-11 所示左下角的软件结构中，主模块为"选择并处理"；如果存在输入部分，则需要设计一个输入模块，在图中左下角的软件结构中，输入部分为"选择服务"；最后为每个活动路径设计一个事务处理的模块，在图中左下角的软件结构中，模块"取款处理"、"转账处理"、"余额查询处理"、"明细查询处理"、"修改密码"对应了各个活动路径；如果各活动路径又集中到一个加工，则需要设计输出模块。

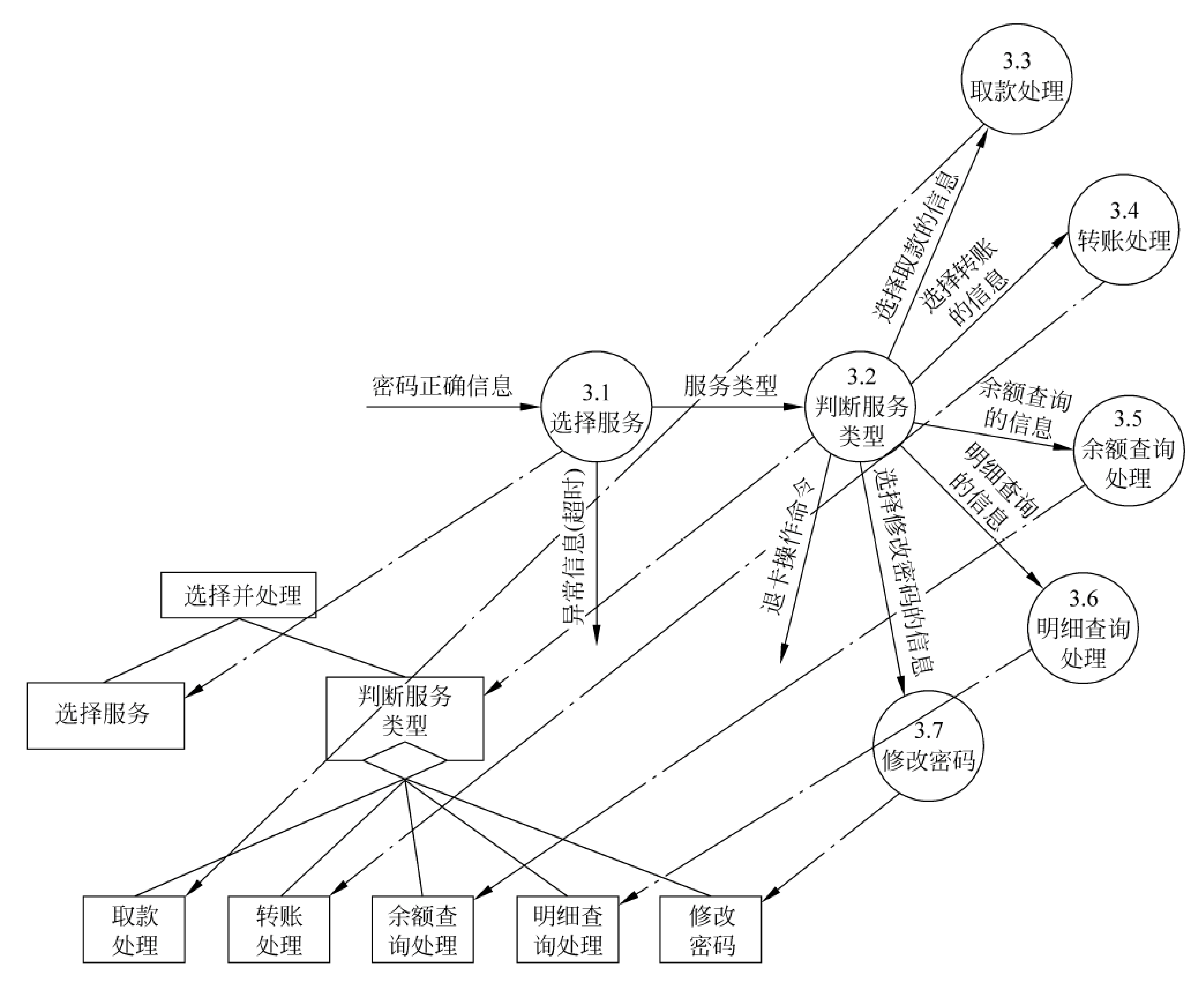


图 8-11　ATM 系统中的事务分析

第三步：对每个模块进行进一步的细化。

对于下属模块的分解细化，也需要根据信息流的类别来判断是采用变换分析还是事务分析。

通过上面提及的从数据流图到最终的软件体系结构的分析设计过程，以及介绍的变换分析方法和事务分析方法，进行多次细化和精化，就可以分析出系统的软件体系结构，如图 8-12 所示。

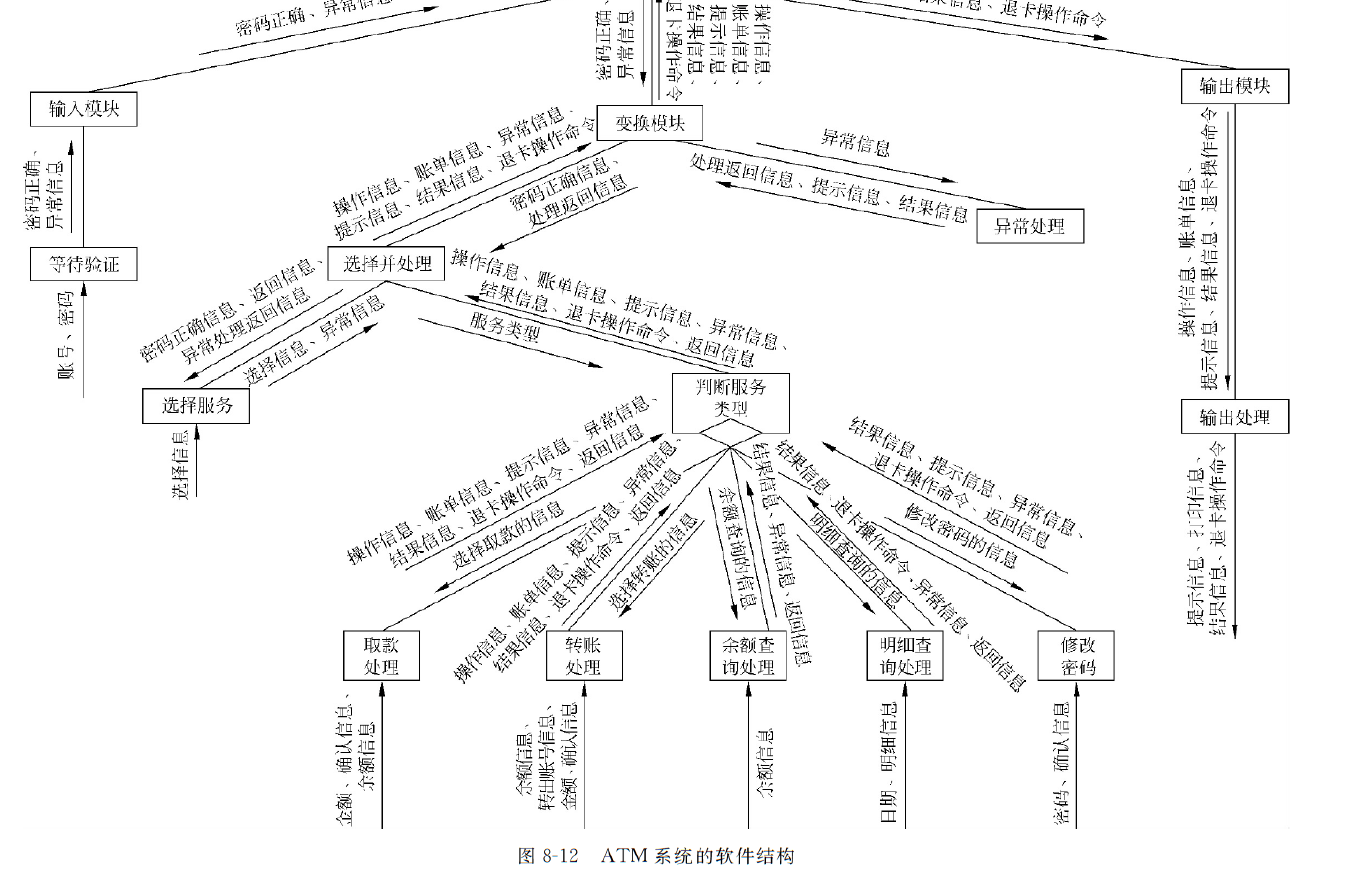


图 8-12 ATM 系统的软件结构

8.4 过程设计

在计算机技术发展的初期，由于计算机硬件条件受到限制，对运算速度与存储空间都有要求，导致开发人员为了追求高效率，把程序的可理解性、可扩充性等因素放到第二位。

随着计算机应用规模越来越大，应用和开发越来越广泛，计算机硬件与通信技术得到了高速发展，程序设计不再是一两个程序员可以完成的任务。编写程序不能再以片面追求高效率为第一要求，而要综合考虑程序的可靠性、可扩充性、可重用性和可理解性等因素。正是这种发展刺激了程序设计方法与程序设计语言的发展。

Fortran、Cobol、Algol、Basic 等语言是较早期出现的高级程序设计语言。在这个时期没有固定的程序设计方法，程序员由于追求程序的高效率，过分依赖技巧与天分，不太注重所编写程序的结构。存在的一个突出问题就是程序可以进行随意跳转，也就是不加限制地使用 goto 语句，这样的程序对别人来说是难以理解的，程序员自己也难以修改程序。

在软件的生命周期里，设计测试方案和计划、对程序进行维护(修改错误和改进功能)都要求程序员要先读懂程序。对于一个大型软件系统来说，程序员读懂程序的时间可能比写程序的时间要长得多。在计算机软件开发的行业中，人员的流动、岗位变迁现象是一种常态，项目在开发过程中，项目组人员组成可能会发生变动，老的开发人员离开，新的开发人员补充进项目组，就需要花大量的时间先读懂已有的程序，才能谈得上继续开发、修改和改进。因此，衡量程序的好坏，除了要看程序的功能、性能是否满足需求，逻辑是否正确，更重要的是要看程序是否容易被读懂和理解。过程设计的主要任务是要设计出程序的纲要，之后才是编码人员根据这个纲要写出具体的程序代码。过程设计的目标除了要使开发出来的程序满足功能、性能上的要求，同时也要满足代码简洁明了易懂的要求。过程设计的好坏决定了最终代码的质量。随着程序规模与复杂性的不断增长，人们探索出新的程序设计方法。专家证明了只用顺序、选择、循环这 3 种基本控制结构，即可实现任何单入口/单出口的程序。

1972 年 IBM 公司的 Mills 提出程序应该只有一个入口和一个出口。Edsger Wybe Dijkstra① 建议从一切高级语言中取消 goto 语句。于是结构设计方法就这样诞生了。由 Niklaus Wirth② 设计出来的 Pascal 语言，就是根据结构设计方法开发出来的语言。其特点是提炼出程序设计共同的特征并能将这些特征编译成高效的代码，因而成为结构设计的有力工具。C 语言也是一种广为流行的结构化程序设计语言，它具有灵活方便、目标代码效率高、可移植性好等优点。

实践证明，结构设计策略确实减少了程序的出错率，提高了程序执行效率，从而大大减少了维护费用。但到底什么是结构程序设计呢？结构设计是组织和编写正确且易读的程序的软件技术，是一种进行程序设计的原则和方法，按照这种原则和方法可设计出结构清晰、容易理解、容易修改、容易验证的程序。结构设计的目标在于使程序具有一个合理结构，以保证和验证程序的正确性，从而开发出易懂易维护的合理程序。

结构设计包含 3 种基本控制结构：顺序结构、选择结构和循环结构。这 3 种结构都是单入口、单出口的程序结构。事实证明，一个任意大且复杂的程序总能转换成这 3 种标准形

---

① Edsger Wybe Dijkstra(1930—2002)，伟大的荷兰计算机科学家。计算机先驱之一，他开发了程序设计的框架结构。

② Niklaus Wirth 是苏黎世联邦理工学院(Swiss Federal Institute of Technology，ETH)教授，是著名的语言设计者之一，他提出了“数据结构＋算法＝程序”这一著名公式，发明了多种影响深远的程序设计语言，于 1984 年获得图灵奖。

式的组合。以下是几种结构设计的工具。

8.4.1 程序流程图

程序流程图历史悠久，应用广泛，它一直是程序算法设计的主要工具。

程序流程图是人们对解决问题的方法、思路或算法的一种描述。它的优点在于：采用简单规范的符号，画法简单；结构清晰，逻辑性强；便于描述，容易理解。

流程图采用如图 8-13 所示的一些基本符号。

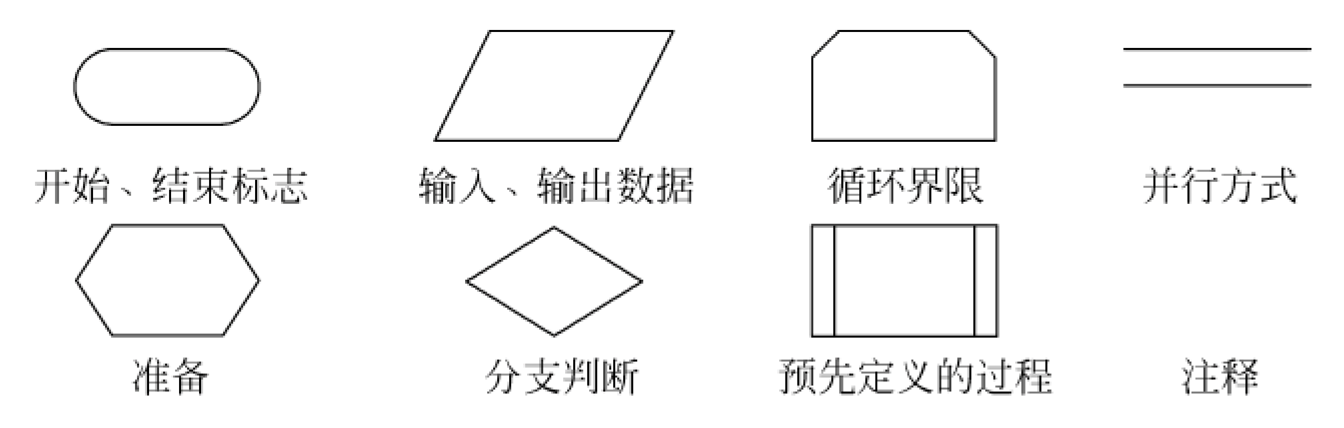


图 8-13 流程图基本符号

例如，使用程序流程图判断某个整数 x 是否为质数的算法，其设计结果如图 8-14 所示。

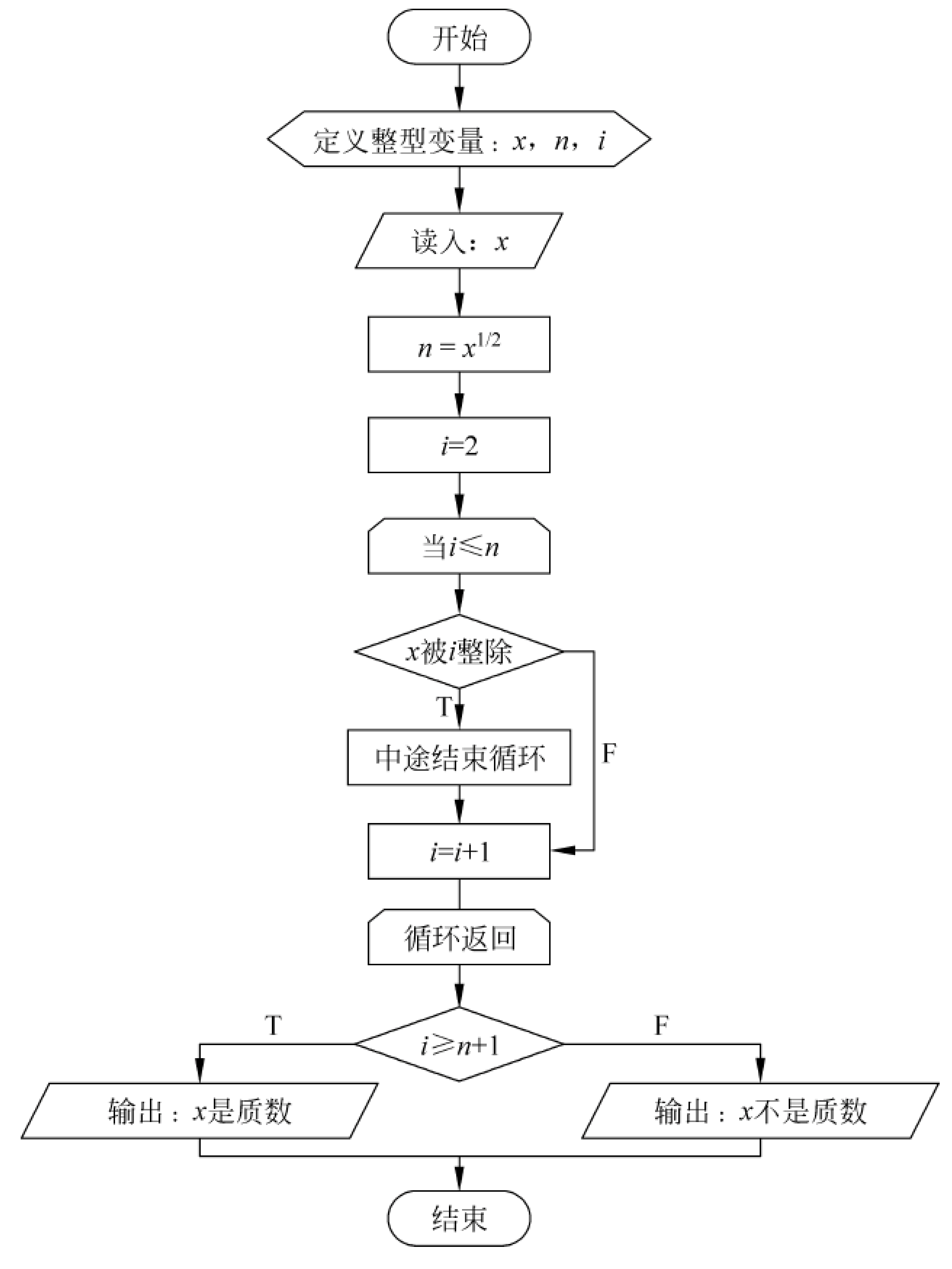


图 8-14 使用程序流程图设计程序算法①

① 曾强聪. 软件工程. 北京：高等教育出版社，2004

传统的程序流程图是一种非结构化的程序算法设计工具，它有以下一些缺点：它无法对嵌套进行清晰的表达，尤其当嵌套比较复杂时；程序流程图无法制止 goto 语句；程序流程图会使程序员过早地考虑程序的控制流程，它不是逐步求精的好工具。

8.4.2　盒式(N-S)图

流程图由一些特定意义的图形、流程线及简要的文字说明构成，它能清晰明确地表示程序的运行过程。在使用过程中，人们发现流程线不一定是必需的，为了不违背结构化程序设计的原则，Nassi 和 Shneiderman 提出了盒式图，它把整个程序写在一个大框图内，这个大框图由若干个小的基本框图构成，在盒式图中，有 3 种基本控制结构的 N-S 图，如图 8-15 所示。

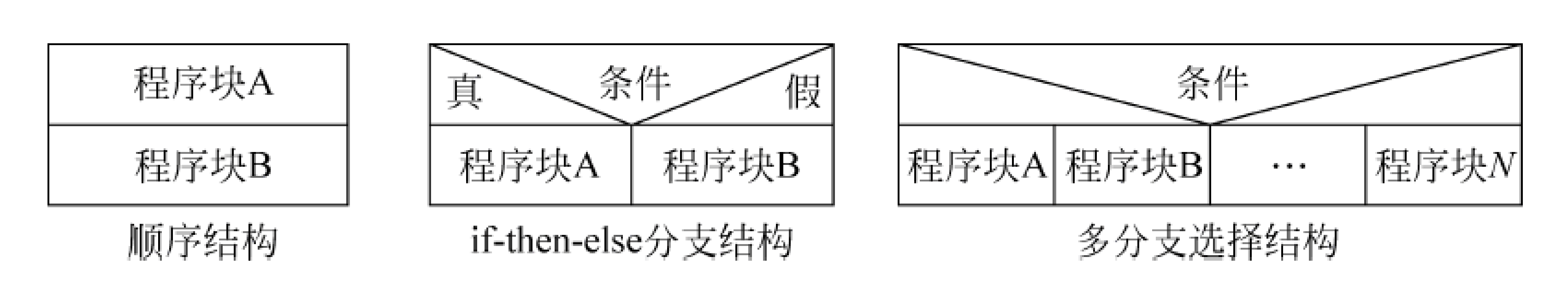


图 8-15　N-S 图基本符号

图 8-16 为判断某个整数 x 是否为质数的算法设计图。

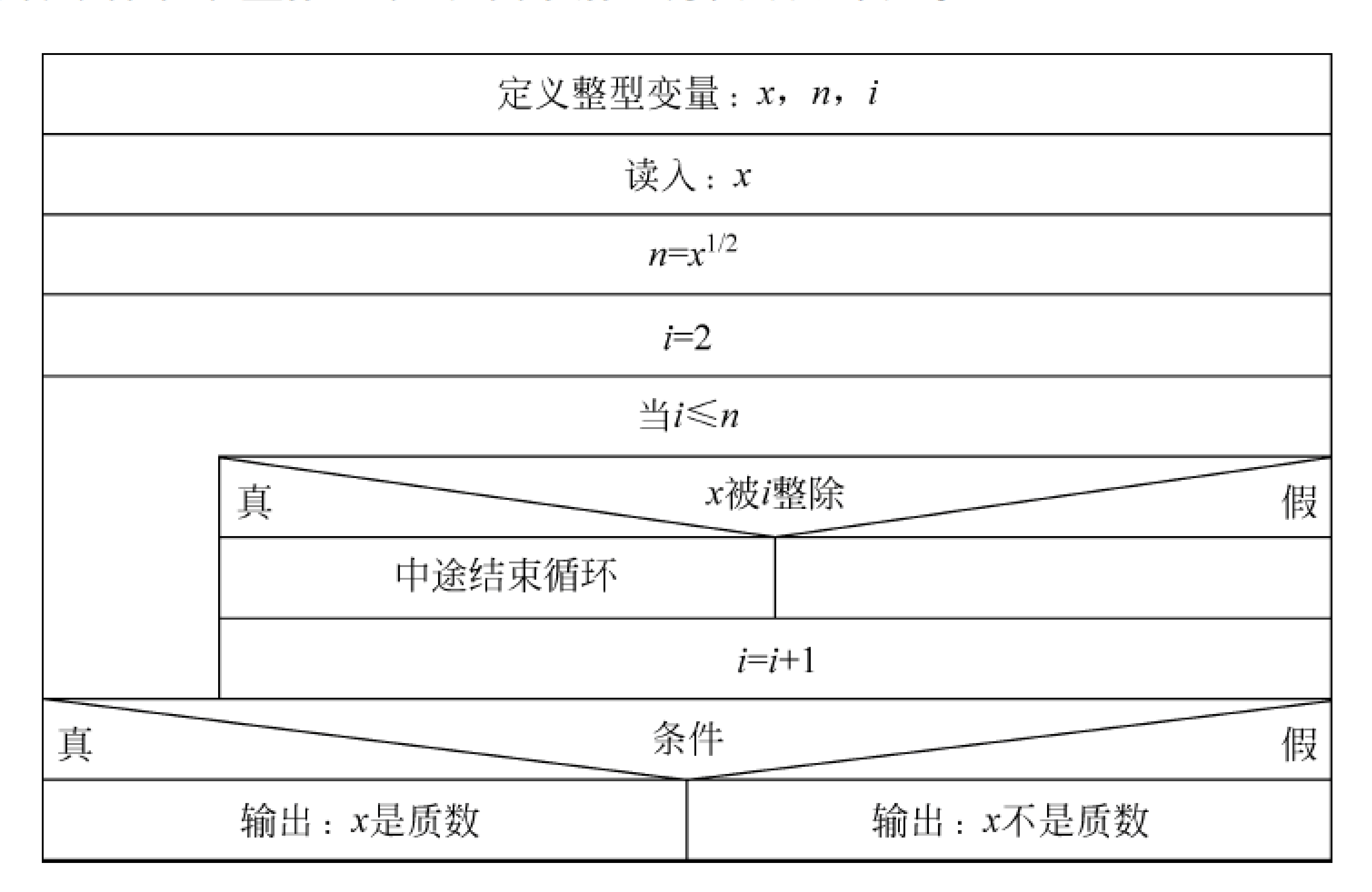


图 8-16　使用 N-S 图设计程序算法

N-S 图实际上是程序流程图去掉控制流线的变种，它有以下特点：①功能表达明确；②容易确定局部数据和全局数据的作用域；③容易表达模块的层次与嵌套关系；④容易培养程序员养成结构化分析问题和解决问题的习惯。

N-S 图的缺点在于，获得结构严密的同时，牺牲了一定的灵活性，不便于进行算法的调整优化，如果问题较为复杂，作图的难度会加大。

8.4.3 PAD

问题分析图(Problem Analysis Diagram,PAD)由日本日立公司于1973年发明,它用二维树形结构的图表示程序的控制流,将这种图转换为程序代码比较容易。

图8-17是PAD的基本符号。

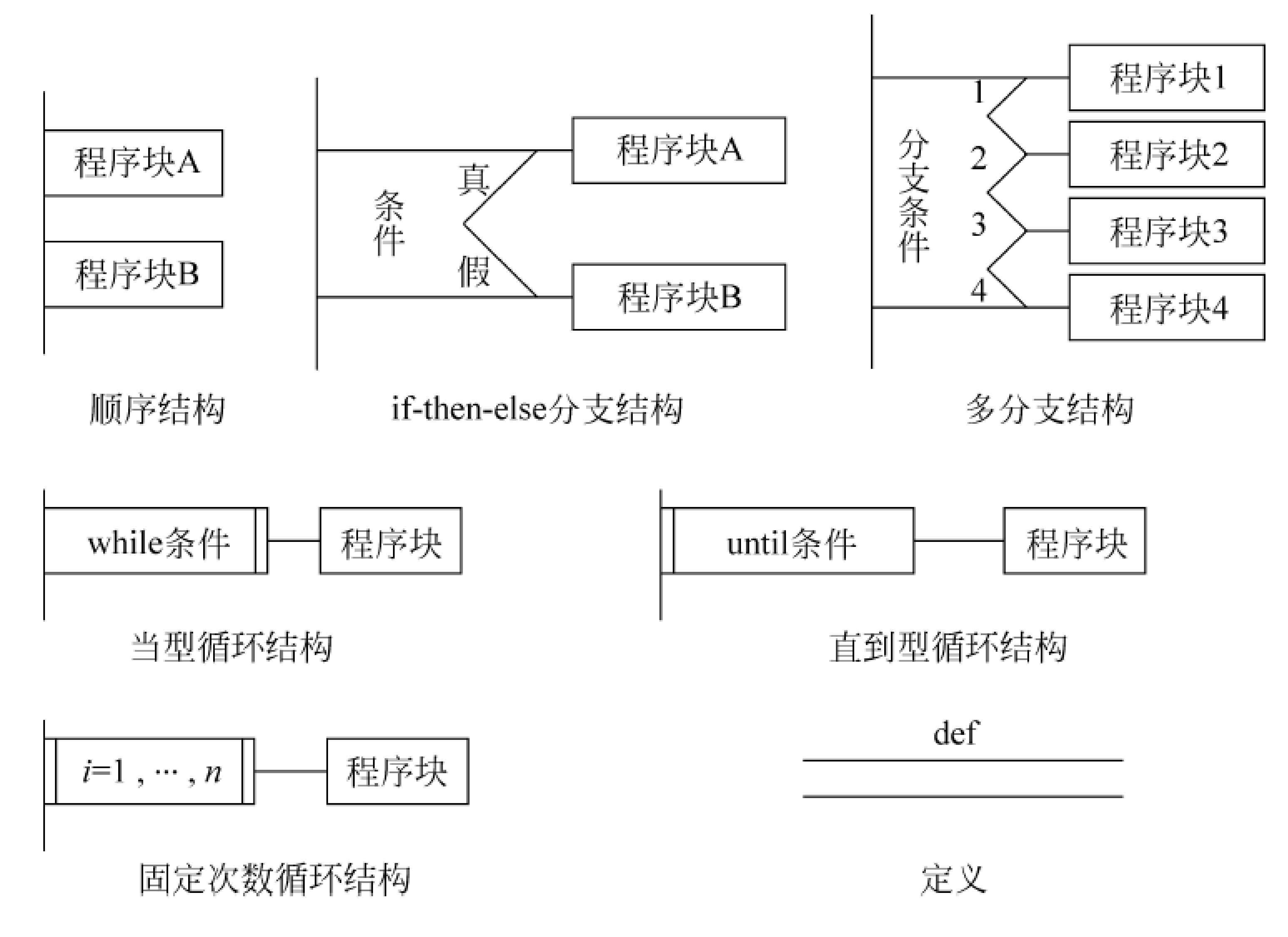


图8-17 PAD基本符号

同样是上面那个判定质数的例子,用PAD表示如图8-18所示。

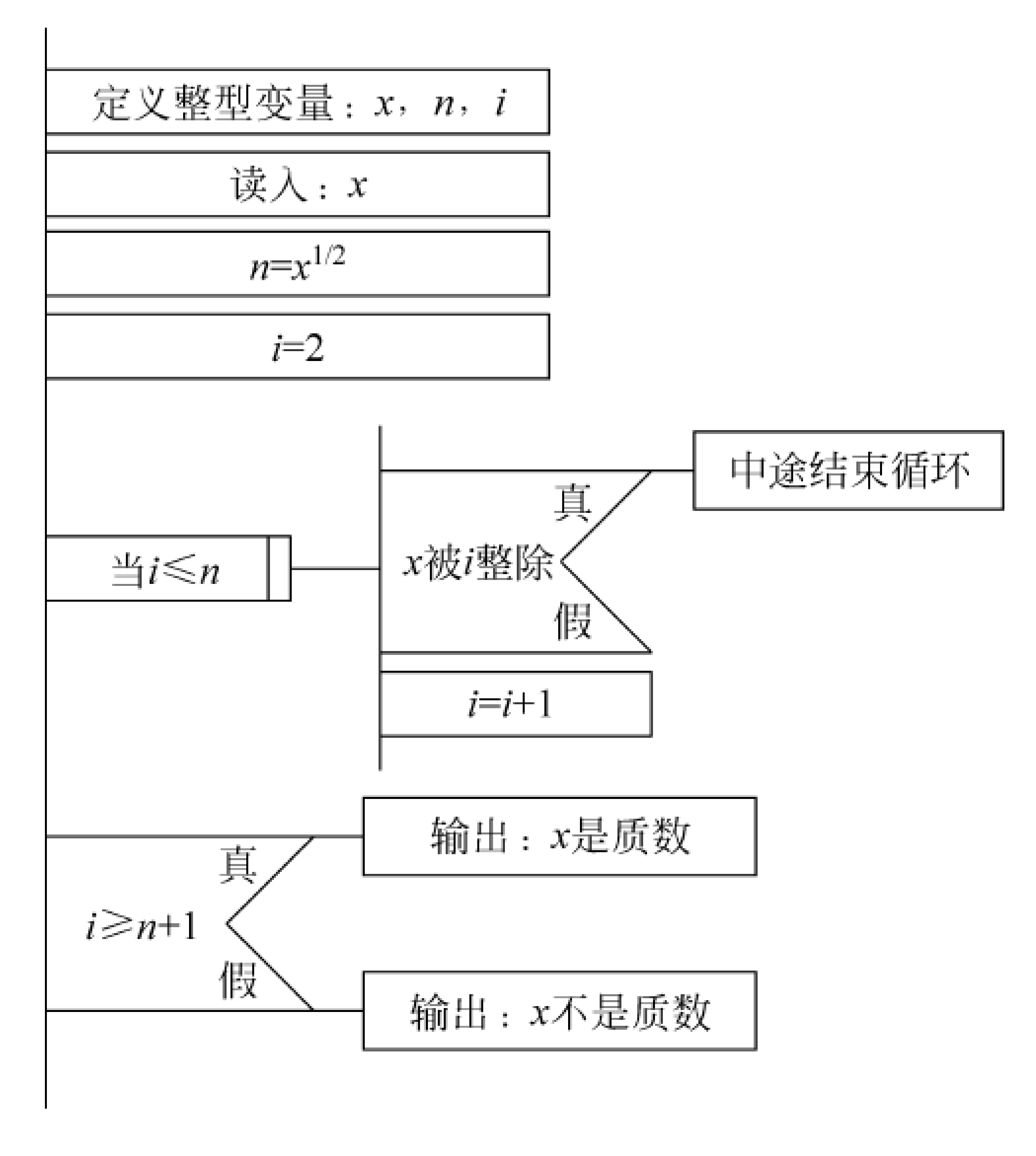


图8-18 使用PAD设计程序算法

PAD 的优点在于：用 PAD 可以容易读懂程序所要表达的逻辑，它是二维树形结构的图形，程序从图中最左边上端的结点开始执行，自上而下，从左到右顺序执行；PAD 是一种程序结构可见性好、结构唯一、易于编制、易于检查和易于修改的详细设计表现方法，用 PAD 可以消除软件开发过程中设计与制作的分离，也可消除制作过程中的主观性；PAD 所描述的程序结构十分清晰。图中最左边的竖线是程序的主线，即第一层控制结构，随着程序层次的增加，PAD 逐渐向右延伸，每增加一个层次，图形向右扩展一条竖线，PAD 中竖线的总条数就是程序的层次数；既可用于表示程序逻辑，也可用于描述数据结构。

8.4.4　PDL

PDL(Program Design Language)也称为伪码，它是用正文形式表示数据和处理过程的设计工具。

PDL 是由 Came、Father 和 Gordon 共同开发的，在 1975 年发表之后，曾进行过重大修改。PDL 具有严格的关键字外部语法，用于定义控制结构和数据结构；另一方面，PDL 表示实际操作和条件的内部语法通常又是灵活自由的，以便可以适应各种工程项目的需要。因此，一般说来 PDL 是一种“混杂”语言，它使用一种语言(通常是某种自然语言)的词汇，同时却使用另一种语言(某种结构化的程序设计语言)的语法。

表 8-2 是 PDL 程序的构成。

表 8-2　PDL 程序的构成类型

PDL 程序的构成类型	子类型	
数据说明		declare <数据名> as <限定词> <限定词>具体的数据结构： scalar <简单变量> array <数组> list <列表> char <字符> structure <结构>
子程序结构		procedure <子程序名> interface <参数表> <分程序 PDL 语句> return end <子程序名>
分程序结构		begin <分程序名> < PDL 语句> end <分程序名>

续表

PDL 程序的构成类型	子类型	
顺序结构	选择型	①if <条件> then <PDL 语句> else <PDL 语句> end if ②if <条件> then <PDL 语句> else if <条件> then <PDL 语句> else <PDL 语句> end if
	WHILE 循环型	loop while <条件> <PDL 语句> end loop
	UNTIL 循环型	loop until <条件> <PDL 语句> end loop
	CASE 型	case <选择句子> of <标号>{,<标号>}: ><PDL 语言> [default]: [<PDL 语句>] end case

同样是上面那个判定质数的例子,用 PDL 表示如下:

```
procedure 判定质数
    declare x,n,j as 整型简单变量
    从键盘读入 x
    将变量 n 赋值为 x^(1/2)
    将变量 i 赋值为 2
    loop while i<=n
        if x 被 i 整除
            中途结束循环
        end if
        变量 i 加 1
    end loop
    if i>=n+1
        输出: x 是质数
    else
        输出: x 不是质数
    end if
end 判定质数
```

PDL 具有下述特点:①它使用关键字的固定语法,为了使结构清晰和可读性好,通常在所有可能嵌套使用的控制结构的头和尾都有关键字,例如,if…end if、loop…end loop 等;

②它使用自然语言的自由语法，来描述具体的处理逻辑；③它需要对数据进行说明，应该既包括简单的数据结构(例如简单变量和数组)，又包括复杂的数据结构(例如，链表或层次的数据结构)；④它需要对模块定义和调用的技术进行说明。

8.4.5 判定表

判定表(Decision Table)是指一个表格，用于显示条件和条件导致动作的集合。判定表是分析和表达多逻辑条件下执行不同操作的情况的工具。如果数据流图的加工需要依赖于多个逻辑条件的取值，使用判定表来描述比较合适。

一张判定表通常由 4 部分组成，左上部列出的是所有的条件，左下部为所有可能的操作，右上部分表示各种条件组合的一个矩阵，右下部分是对应于每种条件组合应有的操作。

以学生的奖学金评定为例，说明判定表的应用。奖励的目的在于鼓励品学兼优的学生，此处理功能是要合理确定奖学金评定等级。决定受奖的条件为：成绩优秀占 70%或 50%以上，成绩为中或中以下占 15%或 20%以下，团结纪律为优或中。奖励方案为一等奖、二等奖、三等奖 3 种。因为受奖条件有些是相容的，相互组合的项较多。描述此学生奖励政策的判定表如表 8-3 所示。

表 8-3 学生的奖学金评定判定表

条件	各门功课的成绩等级比率	优秀≥70%	√	√	√	√	×	×	×	×
		优秀≥50%	—	—	—	—	√	√	√	√
		中以下≤15%	√	√	×	×	√	√	×	×
		中以下≤20%	—	—	√	√	—	—	√	√
	综合素质评分	优	√	×	√	×	√	×	√	×
		中	×	√	×	√	×	√	×	√
奖励方案	一等奖		√							
	二等奖			√	√		√			
	三等奖					√		√	√	√

判定表的优点是它能够将复杂的问题按照各种可能的情况全部列举出来，简明并避免遗漏。因此，利用判定表能够设计出完整的测试用例集合。

在一些数据处理问题中，某些操作的实施依赖于多个逻辑条件的组合，即针对不同逻辑条件的组合值，分别执行不同的操作。判定表很适合于处理这类问题。

8.4.6 判定树

判定树是判定表的变种，也能清晰地表示复杂的条件组合与应做的动作之间的对应关系。判定表虽然能清晰地表示复杂的条件组合与应做的动作之间的对应关系，但其含义却不是一眼就能看出来的，初次接触这种工具的人要理解它需要有一个简短的学习过程。此外，当数据元素的值多于两个时，判定表的简洁程度也将下降。

图 8-19 是一个软件工程组织开始某系统前分析出的一个决策树，从决策树可以看出，该系统可以通过构造(从头开始构造系统)、复用(复用已有的“具有部分经验”的构件来构造系统)、购买(购买现成的软件产品，并进行修改以满足当前项目的需要)、承包(将该系统承

包给外面的开发商)4 种方式获得。其中,就构造方式来看,估计这个系统有 30%的工作是简单的,有 70%的工作是困难的,同时如果构造一个系统的工作难度都是简单的,则完成这个系统的成本是￥380 000,同时如果构造一个系统的工作难度都是困难的,则完成这个系统的成本是￥450 000。

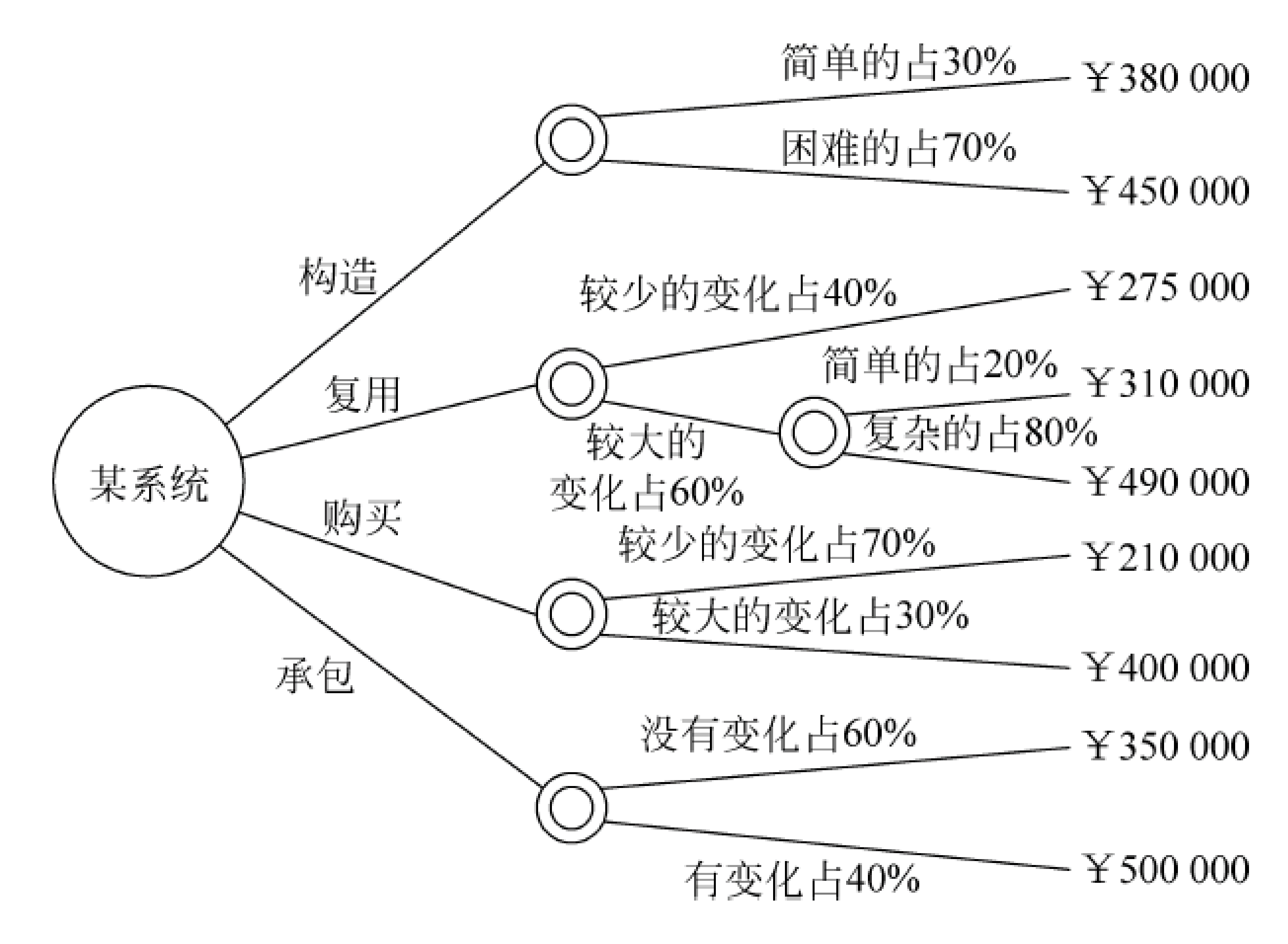


图 8-19　一个支持自行开发、购买决策的决策树

判定树易于掌握和使用,它形式简单,不需任何说明,一眼就可以看出其含义。从不足方面来看,判定树虽然比判定表直观,但简洁性却不如判定表,数据元素的同一个值往往要重复画出多遍,而且越接近树的叶端重复次数越多。此外还可以看出,画判定树时分枝的次序可能对最终画出的判定树的简洁程度有较大影响。

8.5　Jackson 设计方法

M. A. Jackson 提出了一类至今仍广泛使用的软件开发方法,该方法有时也称为面向数据结构的软件设计方法。这一方法从目标系统的输入、输出数据结构入手,导出程序框架结构,再补充其他细节,就可得到完整的软件体系结构。对于输入、输出数据结构明确的中小型系统,这个方法比较有效。Jackson 设计方法也可与其他方法结合,用于模块的详细设计。

1. Jackson 结构图

Jackson 结构图是 Jackson 方法提供的工具。虽然实际使用的数据结构种类繁多,但是数据元素间的逻辑关系只有顺序、选择和重复 3 类。这里给出了 3 种基本结构的表示。

(1) 顺序结构

顺序结构的数据由一个或多个数据元素组成,每个元素按确定次序出现一次。图 8-20(a)是相应的顺序结构图。

(2) 选择结构

选择结构的数据包含两个或多个数据元素,可以按照选择的条件,选择使用相应的数据元素。图 8-20(b)是相应的选择结构图。

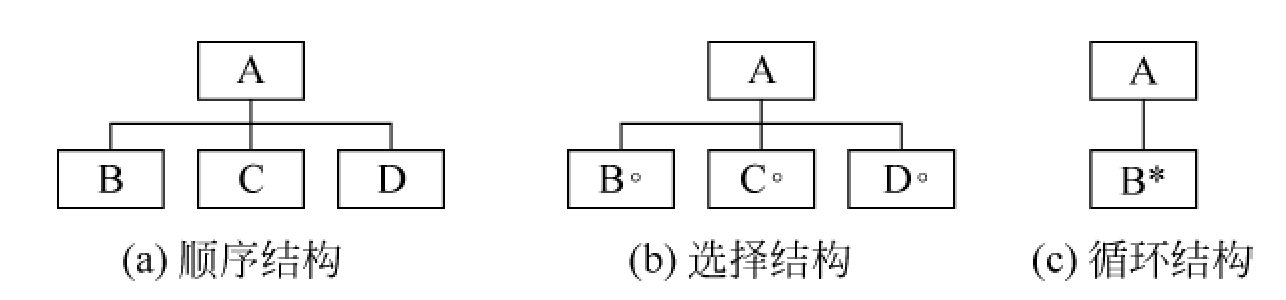


图 8-20　3 类 Jackson 结构图

(3) 重复结构

重复结构的数据，可以根据条件，重复使用零次或多次数据元素。图 8-20(c)是相应的重复结构图。

2. Jackson 程序设计的步骤

一般通过以下 5 个步骤来完成设计。

(1) 分析并确定输入数据和输出数据的逻辑结构，并用 Jackson 结构图来表示这些数据结构。

(2) 找出输入数据结构和输出数据结构中有对应关系的数据单元。

(3) 按以下的规则由输入、输出的数据结构导出程序结构。

① 为每一对在输入数据结构和输出数据结构中有对应关系的单元画一个处理框。

② 为输入和输出数据结构中剩余的数据单元画一个处理框。

③ 所有处理框在程序结构图上的位置，应与由它处理的数据单元在数据结构 Jackson 图上的位置一致。

④ 必要时，可以对映射导出的程序结构图进行进一步的细化。

(4) 列出基本操作与条件，并把它们分配到程序结构图的适当位置。

(5) 用伪码写出程序。

3. Jackson 程序设计举例

假设一个学生成绩管理系统，需要根据学生的每门功课的成绩，计算出该学生的总成绩以及所有学生的平均总成绩，按照学生学号产生学生成绩汇总表。需要输入两个数据表，其中学生信息表为主表，学生成绩记录表为从表，这两个表分别如表 8-4 和表 8-5 所示：

表 8-4　学生信息表

学号	姓名	性别	班级编号
2005814058	张　伟	男	0812
2005814059	李一栋	男	0811
2005814062	王季斌	男	0812

表 8-5　成绩细表

学号	科目	成绩	学号	科目	成绩
2005814058	语文	66	2005814059	政治	91
2005814058	数学	88	2005814062	语文	80
2005814058	政治	95	2005814062	数学	71
2005814059	语文	72	2005814062	政治	60
2005814059	数学	74			

需要产生的学生成绩汇总表如表 8-6 所示。

表 8-6 学生成绩汇总表

学号	姓名	总成绩
2005814058	张伟	249
2005814059	李一栋	237
2005814062	王季斌	211
平均总成绩		232.3

根据 Jackson 方法的设计步骤：

(1) 确定输入输出的数据结构。这个步骤实际上就是一个从数据表输入数据，然后通过报表输出数据的问题。在本例中，输入的数据表分为主表学生信息表和从表成绩细表，图 8-21 是用 Jackson 图绘制的输入输出数据结构，左边为输入的数据结构，右边为输出的数据结构。

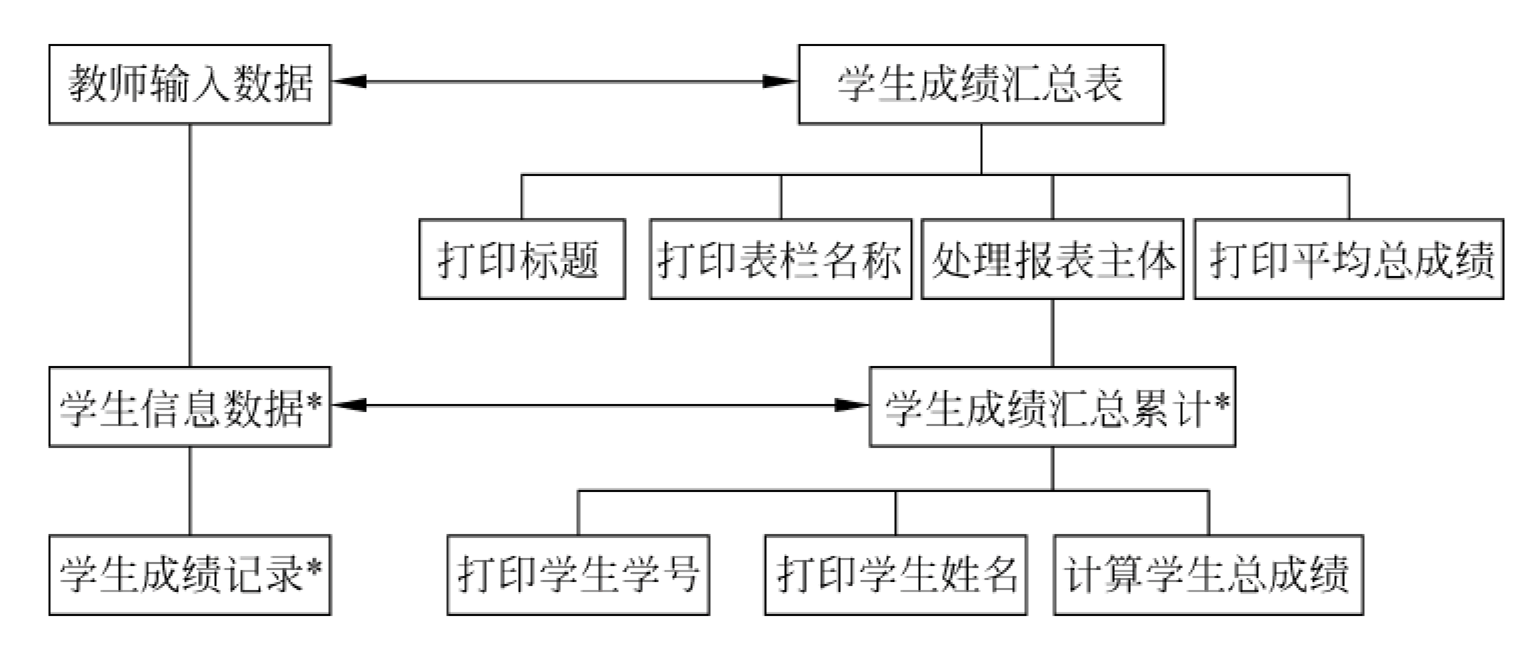


图 8-21 输入输出对应关系结构图

(2) 在确定了输入输出的数据结构以后，要比较两者之间的关系，考虑它们之间的对应关系。输出的数据总是根据输入的数据处理得到的，从图 8-21 可以看出，在输入的最高层次单元“教师输入数据”和输出的最高层次单元“学生成绩汇总表”之间总是存在因果对应关系。此外，由于成绩汇总时的累计单位是学生，所以输入的“学生信息数据”与输出的“学生成绩汇总累计”之间存在着对应关系。

从数据结构图导出到程序基本框架，如图 8-22 所示。根据上面的规则，细分为下面 3 个子步骤。

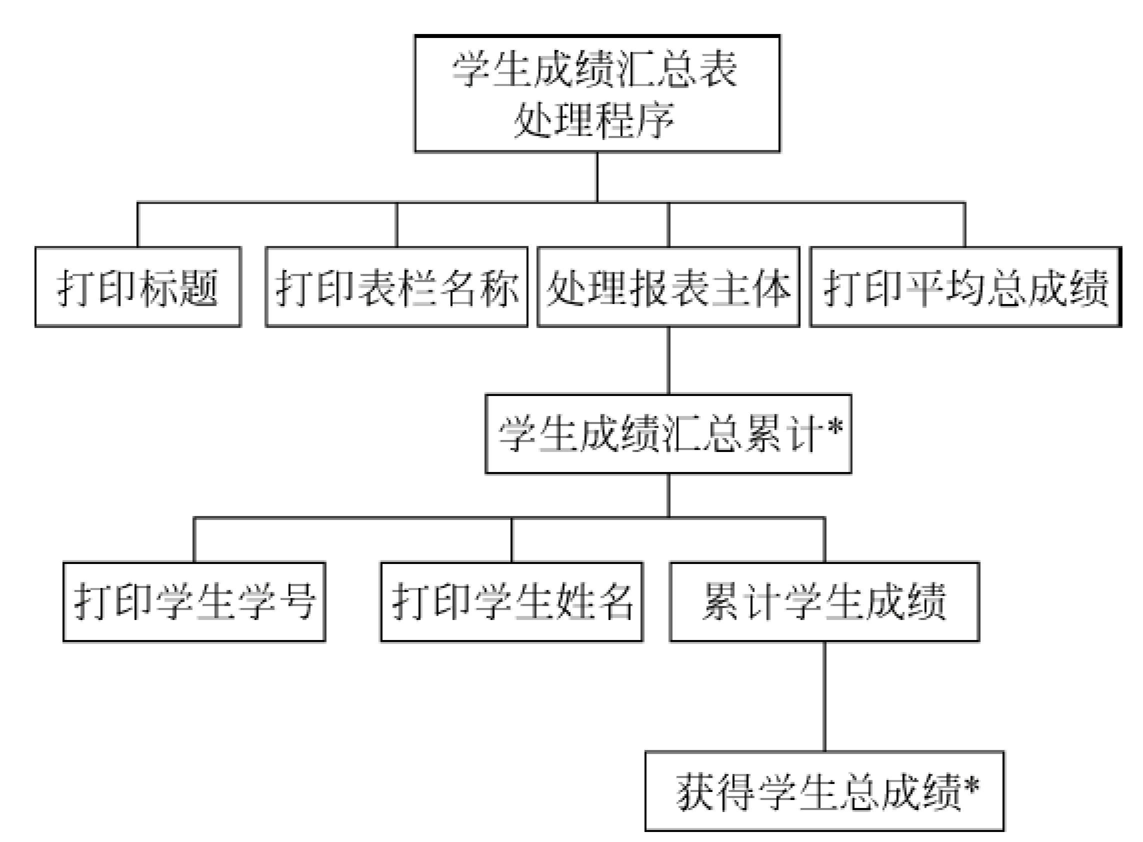


图 8-22 学生成绩汇总程序的 Jackson 程序结构基本框架

为每对有直接对应关系的输入输出数据单元，按照其所对应的层次，在 Jackson 程序结构图中画出一个处理框。例如，程序结构图的最顶层的处理框“学生成绩汇总表处理程序”与输入的“教师输入数据”和输出的“学生成绩汇总表”这对最顶层的数据单元相对应；“以学生信息数据为单位产生总成绩”与输入的“学生信息数据”和输出的“学生成绩汇总累计”这对数据单元相对应。

根据输入输出数据结构中剩余的每个数据单元所处的层次，在程序结构图的相应层次分别为它们画上相对应的处理框。例如，“打印标题”、“打印表栏名称”、“处理报表主体”、“打印平均总成绩”、“打印学生学号”、“打印学生姓名”、“累计学生成绩”、“获得学生总成绩”。

根据上述两步可以得到如图 8-22 所示的程序结构基本框图。

列出基本操作与条件，并把它们分配到程序结构图的适当位置。如图 8-23 所示，这样就可以得到较为完整的 Jackson 程序结构图。

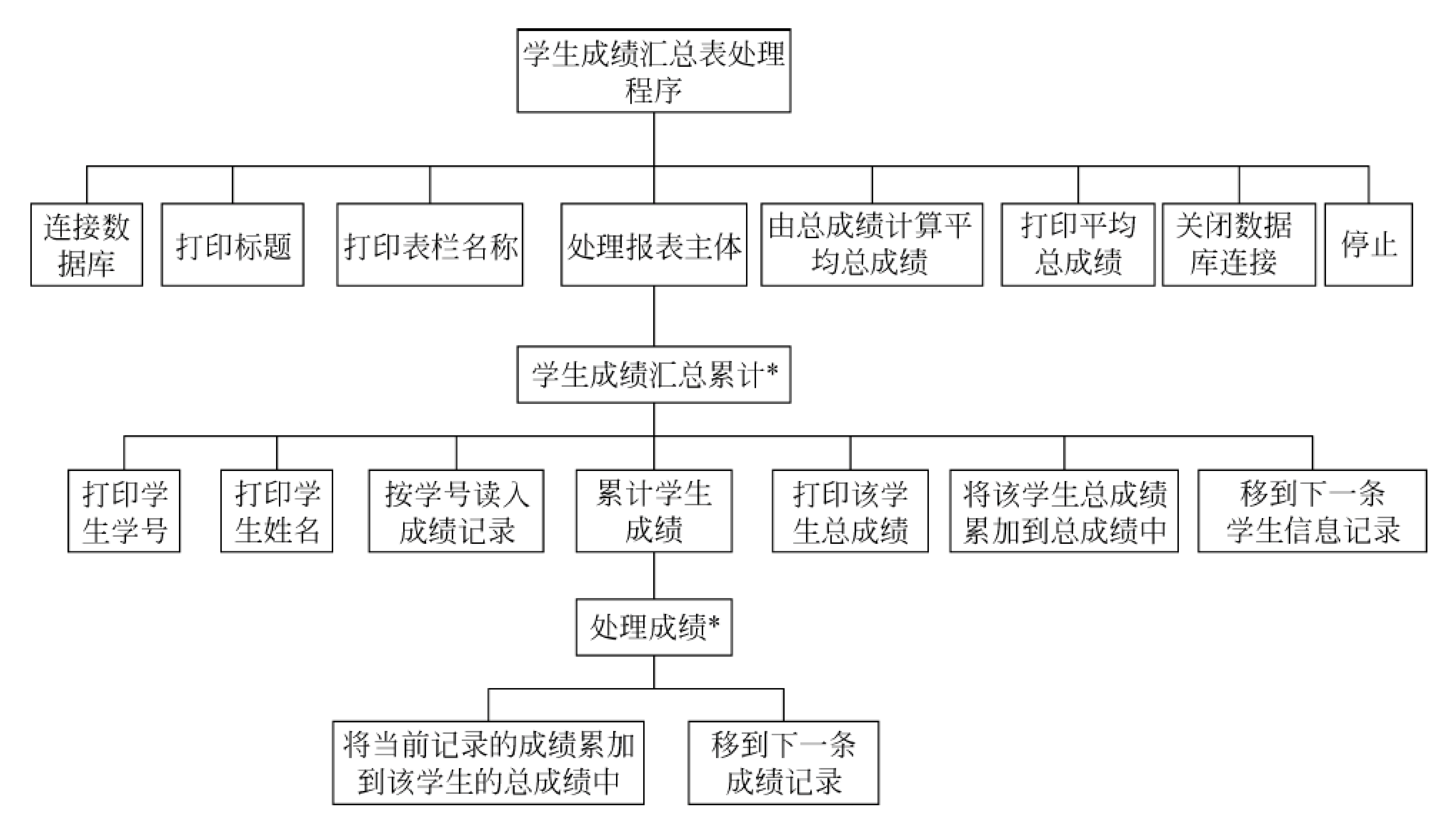


图 8-23　学生成绩汇总程序的 Jackson 程序结构较完整框架

最后一步，就是将 Jackson 程序结构图对应成伪码。因为 Jackson 使用的伪码和 Jackson 结构图之间存在简单的对应关系，所以根据图 8-23 可以很方便地得到以下伪码：

```
procedure 学生成绩汇总表处理程序
    连接数据库
    打印标题
    打印表栏名称
    loop while 学生信息表底
        打印学生学号
        打印学生姓名
        按学号读入成绩记录
        loop while 成绩细表底
            将当前记录的成绩累加到该学生的总成绩中
            移到下一条成绩记录
```

```
            end loop
            打印该学生总成绩
            将该学生总成绩累加到总成绩中
            移到下一条学生信息记录
        end loop
        由总成绩计算平均总成绩
        打印平均总成绩
        关闭数据库连接
    end 学生成绩汇总表处理程序
```

本 章 小 结

抽象与求精、信息隐蔽、模块化设计、模块独立性等有关原理是设计的基础。

抽象是从众多的事物中抽取出共同的、本质性的特征，而舍弃其非本质的特征。信息隐蔽是指每个模块的内部实现细节对外部来说是看不见的。模块化使复杂的大型软件系统能被高效地开发和管理。模块独立是指系统中的模块尽可能地只涉及自己特定的子功能，并且模块接口简单，与其他模块没有过多的通信。一般采用耦合和内聚这两个定性的技术指标来对模块的独立性进行衡量。结构图是描绘软件体系结构的一种工具，它用图形的表示方法来描绘软件的体系结构，它描绘出系统的组织结构和各元素之间的相互关系。

结构化软件设计，是面向数据流的设计方法，数据流图是设计的基础。结构化设计定义了不同的映射，利用这些映射可以把数据流图变换成软件结构。不同的信息流选用不同的映射方法。变换流体现的是数据从输入到加工，再到输出的一般步骤。当输入的信息流可以引发多个不同的事务活动流程，并且数据流图中有一个事务调度中心时，那么称这种信息流为事务流。变换分析就是从变换流分析出软件结构，事务分析是从事务流分析出软件结构。

过程设计的目标是对系统做出精确的设计描述，主要有以下几种可以使用的技术：①程序流程图；②盒式图 N-S；③PAD；④PDL；⑤判定表；⑥判定树；⑦Jackson 设计方法。

思考与练习

1. 储户将填好的存(取)款单、存折交银行，银行工作人员做如下处理：根据存(取)款单选择存款还是取款操作；如果是存款则进行存款处理；如果是取款则进行取款处理；存(取)款后打印存折。图 8-24～图 8-27 是该系统的分层数据流图，试将其转换为软件结构图。

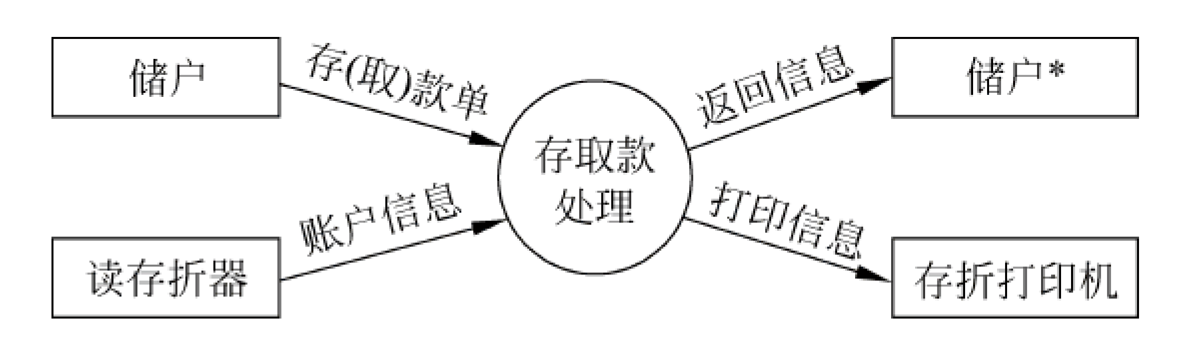


图 8-24 银行存(取)款系统顶层 DFD 图

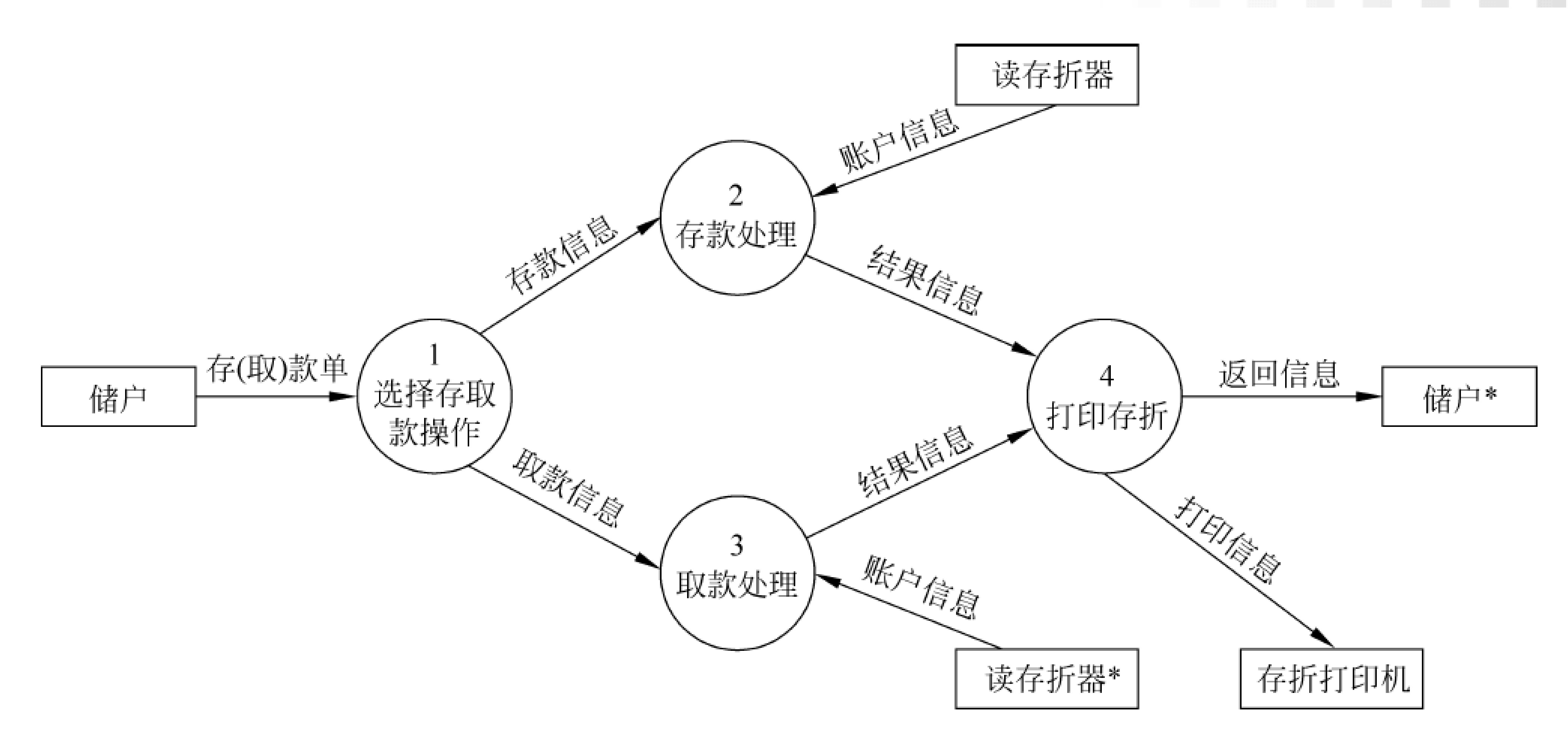


图 8-25　银行存(取)款系统 1 层 DFD 图

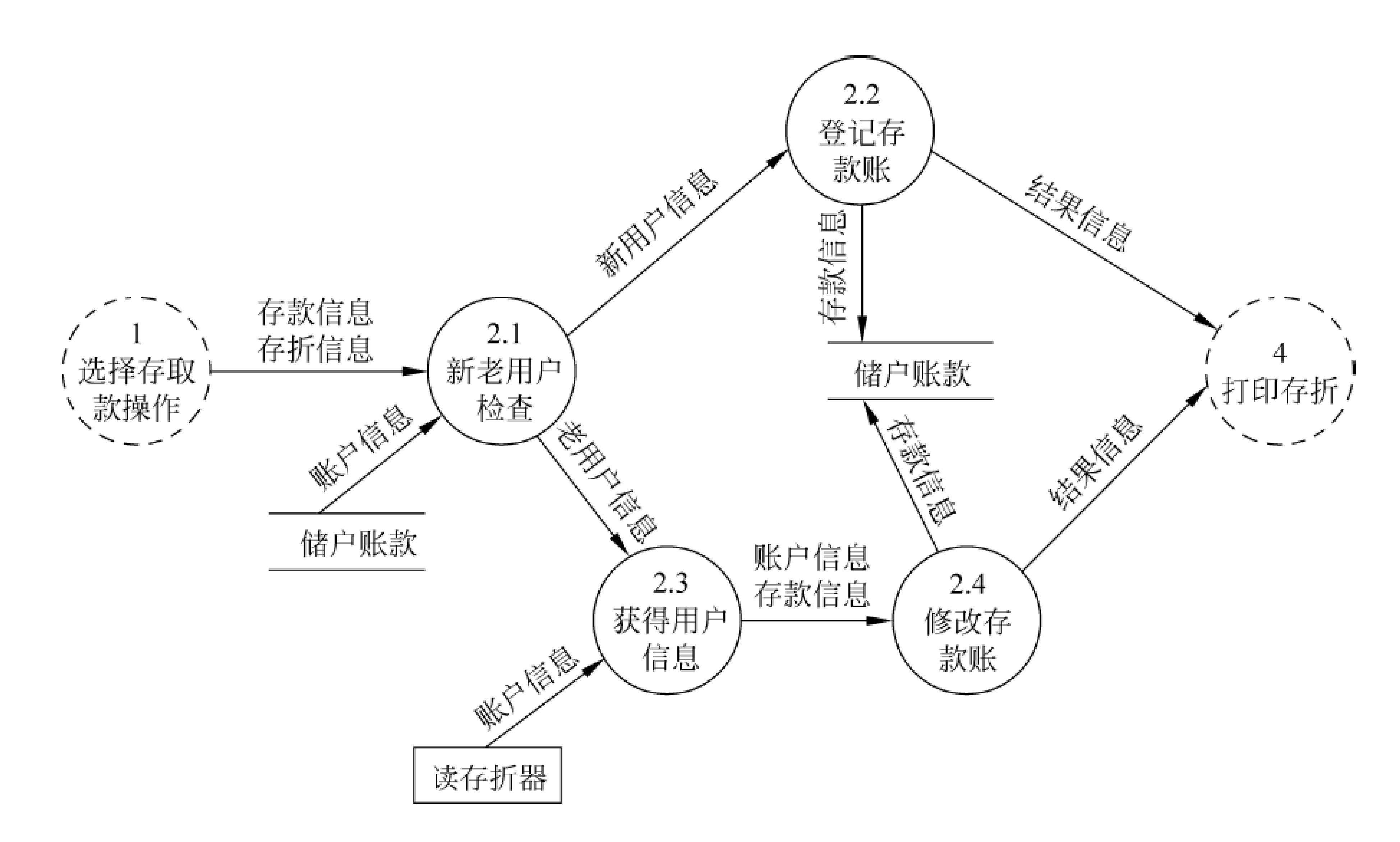


图 8-26　银行存(取)款系统 2 层 DFD 图(“2—存款处理”展开)

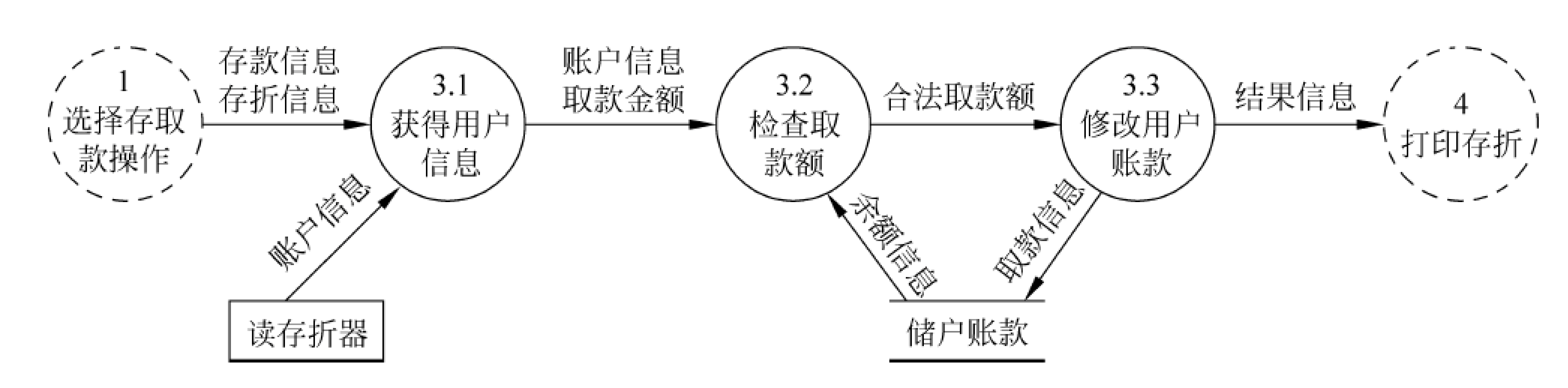


图 8-27　银行存(取)款系统 2 层 DFD 图(“3—取款处理”展开)

2. 将第 7 章思考题 8 的数据流程图转换为软件结构图。

3. 图书馆的预订图书子系统有如下功能：①由供书部门提供书目给订购组；②订购组从各单位取得要订的书目；③根据供书目录和订书书目产生订书文档留底；④将订书信息

(包括数目,数量等)反馈给供书单位;⑤将未订书目通知订书者;⑥对于重复订购的书目由系统自动检查,并把结果反馈给订书者。试根据要求画出该问题的数据流图,并将其转换为软件结构图。

4. 画出下面由 PDL 写出的程序的 PAD(其中 A、B、C1、C2、D1、D2、D3、E、F 表示语句块)。

```
A
loop while a
   B
   if b>0 then
      C1
   else
      C2
   end if
   case d of
      case d1:D1
      case d2:D2
      default:D3
   end case
   E
end loop
F
```

5. 画出第 4 题中由 PDL 写出的程序的 N-S 图。

6. 邮寄包裹收费标准如下:

若收件地点在 1000 千米以内,普通件每千克 2 元,挂号件每千克 3 元。若收件地点在 1000 千米以外,普通件每千克 2.5 元,挂号件每千克 3.5 元。任何包裹若重量大于 30 千克,超出部分每千克加收 0.5 元。请绘制确定收费的判定树和判定表。

第9章 面向过程的结构化实现

程序测试只能表明错误的存在，而不能表明错误不存在。

——Edsger Wybe Dijkstra[①]

9.1 概述

软件实现包括“编写程序”和“测试程序”。“编写程序”是在详细设计的基础上进行的，它将详细设计得到的处理过程的描述转换为基于某种计算机语言的程序，即源程序代码。“测试程序”则是在软件投入运行前，对软件需求分析、设计规格说明和编码的最终复审，是发现软件故障，保证软件质量，提高软件可靠性的主要手段。随着人们对软件质量越来越重视，软件测试在软件开发中的地位也越来越重要。

9.2 编码

9.2.1 软件编码的基本概念

什么是编码的目的？编码的目的就是实现人和计算机的通信，指挥计算机按人的意志正确工作。

什么是编码的任务？编码的任务就是把软件设计转换成计算机可以接受的程序代码。换句话说，也就是写成以某一种程序设计语言表示的“源程序清单”，同时满足结构良好、清晰易读、与设计相一致、具有良好的程序设计风格的要求。

什么是程序设计风格？程序设计风格就是人们在长期的编程实践中形成的一套独特的习惯做法和编程方式。具体要从源程序文档化、数据说明、语句构造、输入和输出以及效率这5个方面来考虑。以下对这5个方面进行进一步解释。

(1) 源程序文档化

有以下但非完全的注意事项：应按照意思为标识符取名字。如果标识符是由多个单词组成的，那么每个单词的第一个字母要大写，或用下划线分开，这样才利于读者理解程序代

---

① Edsger Wybe Dijkstra(1930—2002)，伟大的荷兰计算机科学家。计算机先驱之一，他开发了程序设计的框架结构。

码；另外，应在源程序的适当位置加注释，注释是程序员与读者间通信的重要工具，用自然语言或伪码表示，注释分为序言型注释和功能型注释两类。

（2）数据说明

为使数据易于理解和维护，开发人员在编写源代码时要注意以下指导原则：数据说明顺序要规范，使数据的属性易于查找，从而有利于测试、纠错与维护；当一条语句声明多个变量时，各变量出现顺序要按字典顺序排列；碰到复杂的数据结构，要加注释，说明在编写程序实现时的一些注意要点。

（3）语句构造

语句构造要简单直接，既要讲究效率，又要讲究代码清晰明了，不能为了追求效率而使代码复杂化。注意以下指导原则：避免使用复杂的条件测试；排除测试条件“非”；避免大量的循环嵌套和条件嵌套；使用括号清晰地表达逻辑表达式和算术表达式；利用加空或易读的符号来清晰地表示语句的内容。

在进行语句构造时，要经常问问自己：如果我是个刚接触项目的编程人员，我能看懂这些语句吗？

（4）输入输出

输入格式以及输入操作步骤应尽量简单；程序要能够对输入数据的合法性、有效性进行校验，并报告必要的输入状态信息及错误信息；交互式输入时，应尽可能提供可用的选择和边界值；当程序设计语言有严格的格式要求时，应保持输入格式的一致性；输出数据应表格化、图形化。

（5）效率

效率是一个性能要求，包括对处理时间和存储空间的要求。追求效率要建立在不损害程序可读性或可靠性基础之上，即首先保证程序正确、清晰，再提高程序效率。通过选择良好的数据结构与算法来提高程序效率。

9.2.2 程序设计语言的选择

从程序设计语言的发展历史来说，可以分为低级语言和高级语言两大类，低级语言包括第1代语言的机器语言和第2代语言的汇编语言，这两种语言都依赖于相应的计算机硬件，不同的硬件，其代码指令是不同的。由于汇编语言依赖于硬件体系，且助记符量大比较难记，于是人们又发明了更加易用的高级语言。这种语言的语法和结构更类似普通英文，且由于远离对硬件的直接操作，使一般人经过学习之后都可以编程。高级语言包括第3代程序设计语言和第4代超高级程序设计语言(4GL)。第3代程序设计语言利用类英语的语句和命令，尽量不再指导计算机如何去完成一项操作，如Basic、Cobol和Fortran等。第4代程序设计语言比第3代程序设计语言更像英语但过程更弱，与自然语言非常接近，它兼有过程性和非过程性两重特性，如数据库查询语言、程序生成器等。

图9-1从应用特点和语言内在特点两个不同角度对高级语言进行分类。

如果不是在一些很特殊的情况下，如某单片机不支持高级语言的开发，那么开发人员在进行软件开发时，通常优先考虑高级语言，而不是低级语言（主要是汇编语言）。但具体选用哪种高级语言，就要根据开发厂商和用户的实际情况来选择。以下因素值得考虑：语言自身的特性、软件的应用领域、软件开发的环境、软件开发的方法、算法和数据结构的复杂性、

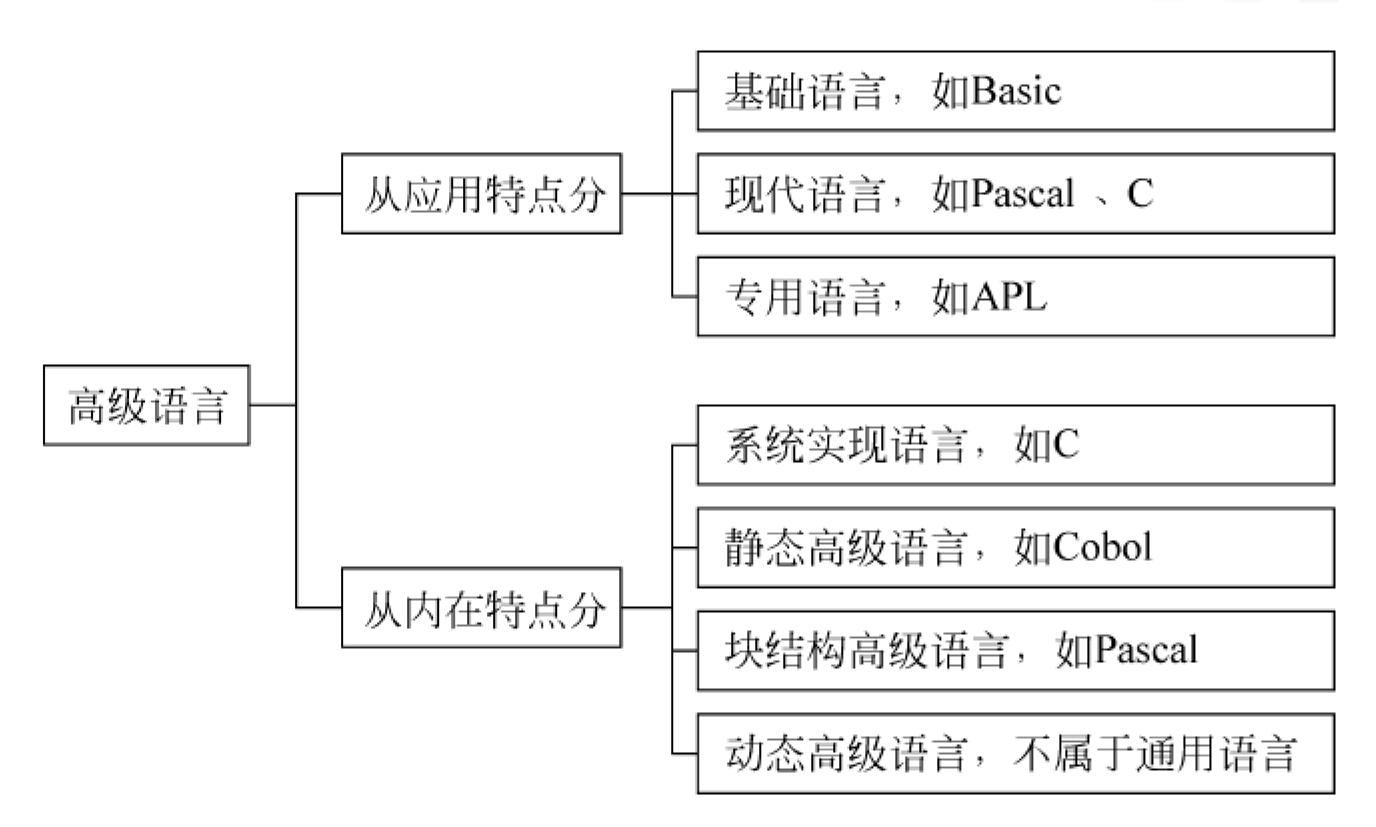


图 9-1 高级语言的分类

软件可移植性要求、软件开发人员的知识等。同时考虑以下选择原则：使程序容易测试和维护以减少软件的总成本，所选用的高级语言应该有理想的模块化机制，以及可读性好的控制结构和数据结构；便于调试和提高软件可靠性，应该使编译程序能够尽可能多地发现程序中的错误；为了降低软件开发和维护的成本，选用的高级语言应该有良好的独立编译机制。

9.2.3 编码风格

很多时候，软件开发人员对自己编写出来的代码毫无自信，有时自己见了都怕，尽管这段代码实现了要求的功能。归其原因，往往是代码风格差导致代码凌乱没有美感。编码风格又称为程序设计风格或编程风格。一个公认的、良好的编程风格可以减少编码的错误，减少读程序的时间，从而提高软件的开发效率。在编码时要善于积累编程经验，培养和学习良好的编程风格，使程序清晰易懂，易于测试与维护，从而提高软件的质量。

9.3 软件测试

9.3.1 软件测试的概述

真正的商用软件程序开发自 20 世纪 50 年代开始发展，从此软件程序的规模经历了爆炸式的增长，发展到现在的千万数量级代码行数，而程序结构和算法复杂度也呈几何级数增长，同时软件开发的协作规模也越来越大，软件不只需要程序员自己能够理解。面对这样的增长态势，如何才能保证程序的正确性和可用性呢？如何在软件程序自身的技术内涵和用户特定领域的需求间找到平衡点？这是一个非常棘手的问题，也是学者和实践者们追寻的目标。软件测试作为度量软件与用户需求间差距的手段登上了历史舞台。

有一些软件中存在的错误虽小，但总是让使用者感到别扭甚至是啼笑皆非，从而对软件产品感到失望。如果你在使用一个软件的过程中，遇到各种各样层出不穷的问题，一定是异常恼火。这些问题出现之后，用户被厂商告知是因为自己的系统无法满足软件的运行需求或设置的问题。有很多软件厂商没有意识到软件的衡量标准中稳定的重要性。软件开发商为了占有市场，必须把产品质量作为企业的重要目标之一。用户希望选用质量优秀的软件

产品。质量不好的软件产品不仅会使维护费用和使用成本大幅度增加，还会造成公司信誉下降。在一些关键的应用领域中，如果产品质量出了问题，还会因此导致一些灾难性的后果。目前，很多人已经认识到：软件中存在的错误导致了软件开发在成本，进度和质量上的失控。由于软件开发的主体是人，人是不可能在所有时间里完全正确的，因此不可能要求开发者开发出的软件是十全十美的，也不可能完全杜绝软件中的错误，但是可以用软件测试的手段使程序中的错误数量尽可能少，密度尽可能小。

专家指出，零缺陷的软件是不存在的。但通过必要的测试，软件缺陷可减少 75%，从而降低软件使用风险。有关机构研究表明，国外软件开发厂商约 40%的工作量要花在测试上，对一些可靠性、安全性要求较高的软件，在测试上要投入更多资源。例如，在 1999 年发布 Windows 2000 操作系统时，微软公司就投入了 250 多个项目经理、1700 多个开发人员，内部测试人员则达到 3200 人，比前两者之和还要多。与此相比，国内软件产业的软件测试人才极其稀缺。由于人才供需失衡，大多数软件厂商测试人员的数量不足开发人员数量的 1/5，远落后于国外先进水平。软件企业测试能力不足的主要原因在于软件测试人才短缺，限制了我国软件产品开发和软件行业发展。对于软件用户来说，软件测试不仅仅应是检验质量的工具，更应成为验证软件产品是否符合用户需求的保障。对软件厂商来说，只有拥有足够的软件测试人才才能对产品进行全面的安全测试，业务才有可能进一步扩展。

那么，到底什么是软件测试呢？

这里引用 Glenford J. Myers 在 *The Art Of Software Testing*① 一书中的观点作为软件测试的定义：软件测试是为了发现缺陷而执行程序的过程，测试是为了证明程序中有错误，而不是证明程序中无错误，一个好的测试用例指的是它可能发现至今尚未发现的缺陷，一次成功的测试指的是发现了新的软件缺陷的测试。

1. 软件测试历史回顾

软件测试的发展总共经历了 3 个阶段：1950—1970 年是软件测试初级时期，在这个阶段是程序编好之后再进行调试，目的是检查程序正确与否；1970—1980 年是软件测试的发展阶段，在这个阶段人们开始重视对测试方法的研究，涌出了大量的测试方法和技术，被人们称为第 2 代软件技术，开始渐渐地用工程化观点来看待测试；1980 年到现在是软件测试成熟时期，测试已逐步形成一套工程方法，发展了很多自动化测试程序，并开始建立测试计划，制定测试方案，设计测试数据，实施测试过程，进行评价分析等。

2. 软件测试的目的

测试的目的是什么？测试的目的是以最少的时间和人力找出软件中潜在的各种错误和缺陷。测试只能尽可能多地查找出程序中的错误，而不能证明程序中没有错误。如果测试是为了证明程序的正确性，那么测试人员往往会设计一些不易暴露错误的测试方案；相反，如果测试是为了发现程序中的错误，测试人员就会力求设计出最能暴露错误的测试方案。因此，正确认识测试的目标对整个测试工作的进行具有指导意义。由于测试的目标是暴露程序中的错误，从心理学角度看，由程序的编写者自己进行测试是不恰当的。因此，在综合

---

① Glenford J. Myers. The Art of Software Testing. John Wiley & Sons, Inc. New York, NY, USA, 1979

测试阶段应该由其他人员组成测试小组来完成测试工作。

另外，软件测试涉及到软件开发周期中各个阶段的错误，它不仅仅是对编码阶段的语法错、语义错、运行错等进行查找的一系列活动，而是对软件计划、软件设计、软件编码进行查错和纠错的活动。纠正过程可能涉及到改正或重新设计相关的文档活动。

3. 测试用例

测试用例目前没有经典的定义。比较通常的说法是：指对一项特定的软件产品进行测试任务的描述，体现测试方案、方法、技术和策略。内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。在测试时将实际的输出与预期结果比较，如果相同则表示没有发现错误，否则表示发现错误。

要使最终用户对软件满意，最有力的举措就是明确阐述最终用户的期望，以便对这些期望进行核实并确认其有效性。测试用例反映了要核实的需求。然而，核实这些需求可能通过不同的方式并由不同的测试员来实施。不同类别的软件，测试用例是不同的。

测试用例是设计和制定测试过程的基础。为了发现程序中的错误应竭力设计不容易使程序通过的测试用例，以及易于暴露程序错误的测试用例。一个好的测试用例自然要考虑那些易于发现程序错误的测试用例。能够高效率揭露至今为止尚未发现错误的测试是成功的测试。测试的深入程度与所设计的有效测试用例的数量成比例。测试用例的数量越多，越能反映不同的场景、条件或经由产品的流程，则越能覆盖基于用户需求的各种功能，对产品质量和测试流程越有信心。通过设计全面且细化的测试用例，可以更准确地估计测试周期各个连续阶段的时间安排。测试设计和开发的类型以及所需的资源主要都受控于测试用例。测试用例通常根据它们所关联的测试类型或测试需求来分类，而且将随测试类型和测试需求进行相应的改变。每个测试需求至少编制两个测试用例：一个测试用例用于证明该测试需求已经满足，通常称作正面测试用例；另一个测试用例反映某个无法接受、反常或意外的条件或数据，用于论证只有在所需条件下才能够满足该需求，这个测试用例称作负面测试用例。

在进行软件测试时，要注意以下一些事项：软件开发人员要避免测试自己开发的程序；在设计测试用例时，要把预期输出的结果作为测试用例的一部分；要测试出程序是否做了该做的任务，还要测试出程序是否做了不应该做的任务；合法的预期的输入数据要编写测试用例，非法的和非预期的输入数据也要编写测试用例；做好保存测试的历史记录，在改错或维护后还要进行重新测试；等等。

9.3.2　软件测试方法

动态测试是软件测试的主要方法，静态测试是不可缺少的辅助方法，而正确性测试可以提供一些研究思路，图 9-2 列出了软件的测试方法。

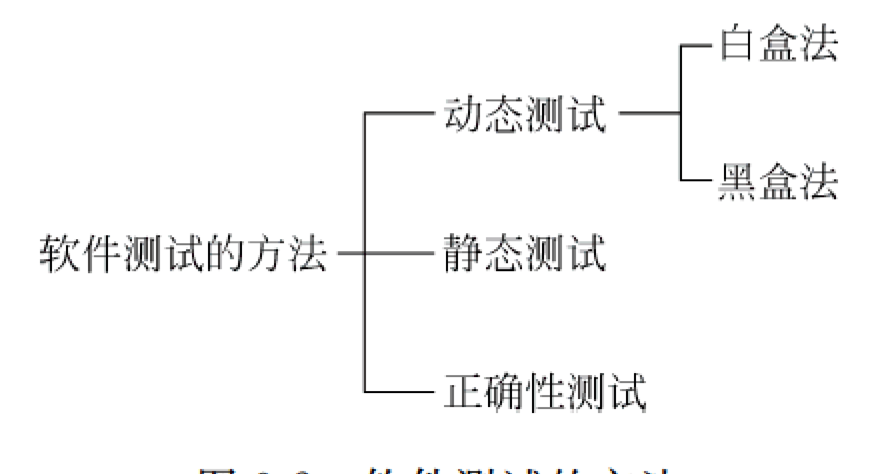


图 9-2　软件测试的方法

虽然 Edsger Wybe Dijkstra 告诉我们“程序测试只能表明错误的存在，而不能表明错误不存在”，然而正确性测试的发明者 Jiri Horejs[①] 建立了一个

① Jiri Horejs(1933—2001)开发了测试程序正确性的方法，并把信息学、计算机科学引入捷克斯洛伐克，因而在 1996 年被 IEEE 授予计算机先驱奖。他在玛莎丽克大学的研究主要就是测试程序正确性的形式化方法。

用于检验程序测试方法的系统，名为 TPT(Test Program-Testing Method)。在这个项目的研究中，他还提出了一种面向图形的程序表达方法，这种方法是基于概念逻辑的(Intentional Logic)，因此在数据库设计和知识表示等领域获得了广泛应用。在测试程序正确性的研究中，他还引入了一种基于语义变化判断程序是否正确的新概念。在这一研究中，他用自动机作为程序内部特性的模型，取得了很好的效果。

静态方法是指不运行被测程序本身，仅通过分析或检查源程序的语法、结构、过程、接口等来检查程序的正确性。根据需求规格说明书、软件设计说明书来找错，以及对源程序进行结构分析和流程图分析来发现问题。静态方法通过程序静态特性的分析，找出欠缺和可疑之处，例如不匹配的参数、不适当的循环嵌套和分支嵌套、不允许的递归、未使用过的变量、空指针的引用和可疑的计算等。静态测试结果可用于进一步的查错，并为测试用例选取提供指导。

动态方法一般是指上机测试，即通过运行被测程序，检查运行结果与预期结果的差异，并分析运行效率和健壮性等性能，这种方法由 3 部分组成：构造测试实例、执行程序、分析程序的输出结果。动态测试可分为两类：白盒测试法和黑盒测试法。

1. 白盒测试法

白盒测试法是把程序装在一个透明的白盒子里，即完全了解程序的结构和处理过程，按照程序内部的逻辑过程，来检验程序的每条通路是否都能按照预定的要求正确工作。

逻辑覆盖是设计白盒测试方案的一种技术，由于不可能对程序进行穷尽测试，选用少量“最有效的”测试数据，有选择地执行程序中的一些通路，做到尽可能完备地测试就更重要了。根据覆盖的程度，由低到高有语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、路径测试。

1) 语句覆盖

语句覆盖就是设计若干个测试用例，运行被测程序，使每一可执行语句至少执行一次。这里的“若干个”，意味着使用测试用例越少越好。语句覆盖率的公式可以表示如下：

$$语句覆盖率=(被评价到的语句数量/可执行的语句总数)\times 100\%$$

图 9-3 是一个用流程图描述程序的处理流程，现在要求用白盒测试法中的“语句覆盖标准”对其进行测试，并要求选取最小的测试数据组。

只需要选取($A=2,B=0,X=4$)这个测试数据组来进行测试，就可以对该程序中的所有语句(a)(b)(c)(d)(e)(f)实现覆盖，语句覆盖对程序的逻辑覆盖很少。语句覆盖是很弱的逻辑覆盖。

2) 判定覆盖

判定覆盖又叫分支覆盖，就是使每个判定的所有可能结果至少出现一次。换句话说，就是所设计的测试用例使程序中的每一个判定取真、取假的值至少经历一次(即判定的每个分支至少经过一次)。

仍以上述流程图为例，要用白盒测试法中的“判定覆盖标准”对其进行测试，并要求选取最小的测试

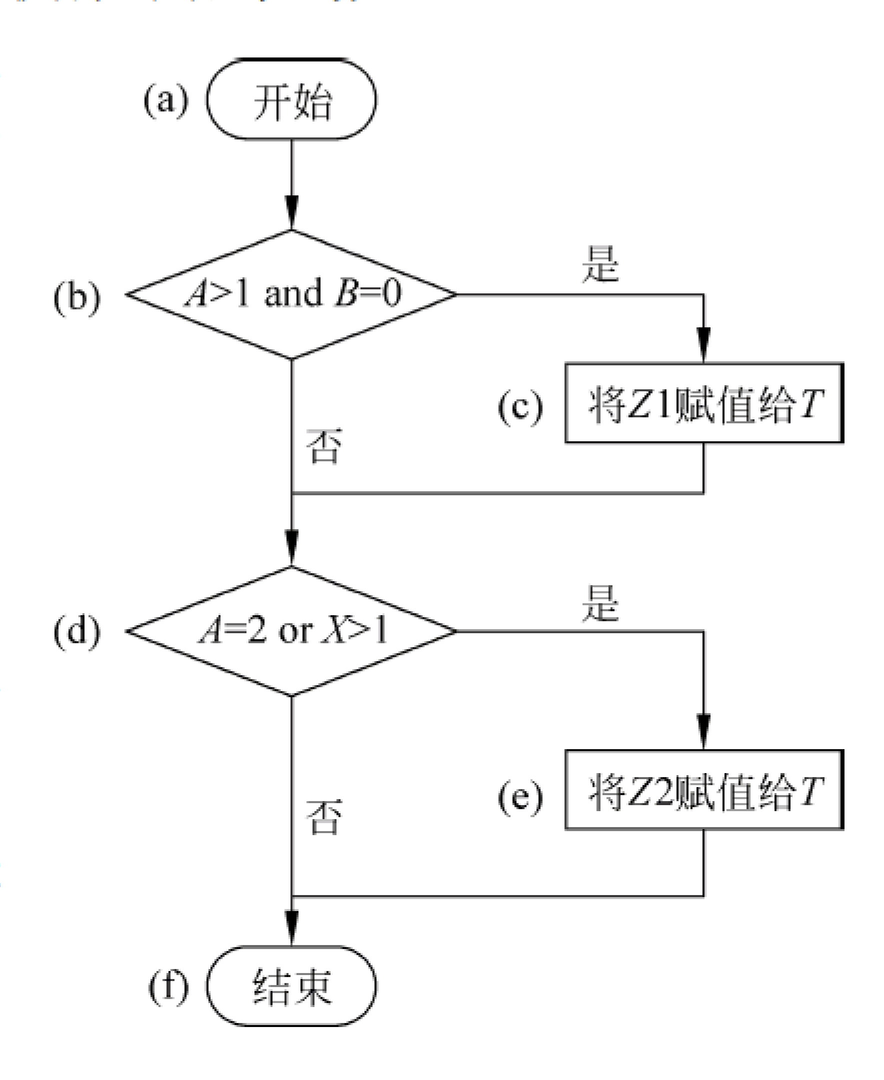


图 9-3　一个程序流程图例子

数据组，程序中共有两个判定“$A>1$ and $B=0$”和“$A=2$ or $X>1$”，如果能够覆盖路径(a)(b)(c)(d)(e)(f)和(a)(b)(d)(f)的话，就可以说实现了判定覆盖。

选取以下两条测试数据进行测试：

(1) $A=3,B=0,X=3$；　　　　该测试数据覆盖(a)(b)(c)(d)(e)(f)

(2) $A=3,B=1,X=0$；　　　　该测试数据覆盖(a)(b)(d)(f)

上述测试数据(1)使得第一个判定为真，第二个判定为真。测试数据(2)使第一个判定为假，第二个判定为假。如果第二个判断条件中 $X>1$ 被程序员错误地写成 $X>0$，使用上述测试用例仍按原路径不影响任何结果。所以可以看出，判定覆盖对程序逻辑的覆盖程度仍然不高。

因此可以得出只做到判定覆盖，仍无法准确判断每个判定的内部条件是否错误，还需要逻辑覆盖更强的方法来检测判定内的条件是否正确。

3) 条件覆盖

一般来说，在软件设计过程中一个判定往往由多个判定条件组成，在上面的例子中，共有两个判定“$A>1$ and $B=0$”和“$A=2$ or $X>1$”，判定一由条件“$A>1$”和“$B=0$”组成；判定二由条件“$A=2$”和“$X>1$”组成。

条件覆盖是指不仅每个判定都取得各种结果，而且判定表达式中的每个条件也都取得各种可能的结果。判定覆盖仅考虑判定的结果而没考虑每个条件的可能性，而条件覆盖却要求每个条件的所有结果至少执行一次。

同样是上面的例子，这次要求用白盒测试法中的“条件覆盖”对其进行测试，并要求选取最小的测试数据组。该程序共有两个判定、4 个条件。选取测试数据时，要使这 4 个条件都可以取到“真”值和“假”值。在使所选取的数据对尽量少的情况下，使每个条件均取一次真值和一次假值，不要出现测试效果完全相同的数据对。要做到条件覆盖，在第一个判定处，$A>1,A\leqslant1,B=0,B<>0$ 都要出现一次；在第二个判定处，$A=2,A<>2,X>1,X\leqslant1$ 都要出现一次。于是选取出以下数据组：

(1) $A=2,B=0,X=4$(满足条件 $A>1,B=0,A=2,X>1$，该测试数据覆盖(a)(b)(c)(d)(e)(f))

(2) $A=1,B=1,X=1$(满足条件 $A\leqslant1,B<>0,A<>2,X\leqslant1$，该测试数据覆盖(a)(b)(d)(f))

这两对数据对该程序判定中条件的执行结果如表 9-1 所示。

表 9-1　满足条件覆盖的测试用例的执行结果

测试用例	$A>1$	$B=0$	$A=2$	$X>1$
(1) $A=2,B=0,X=4$	真	真	真	真
(2) $A=1,B=1,X=1$	假	假	假	假

条件覆盖通常情况下要比判定覆盖强，是否能确定条件覆盖就一定比判定覆盖强？即能否确定条件覆盖在对每个条件进行测试的同时，也会使每个判定都取得所有可能的结果？现在重新选取两组测试数据满足条件覆盖，再来看看其是否可以满足判定覆盖的要求，如表 9-2 所示。

表 9-2 满足条件覆盖却不满足判定覆盖的测试用例

测试用例	条件覆盖				判定覆盖	
	$A>1$	$B=0$	$A=2$	$X>1$	$A>1$ and $B=0$	$A=2$ or $X>1$
(1) $A=2,B=0,X=0$	真	真	真	假	真	真
(2) $A=1,B=1,X=2$	假	假	假	真	假	真
是否满足覆盖	满足				不满足	

从表 9-2 可以看出，新的两组测试数据满足了条件覆盖，却无法满足判定覆盖，即路径(f)未经过。所以得出结论：条件覆盖不一定比判定覆盖强。那么需要什么样的测试数据，可以做到满足条件覆盖，同时又实现判定覆盖呢？答案是：满足判定/条件覆盖的测试用例。

4）判定/条件覆盖

判定/条件覆盖是指选取足够的测试用例，使判定中每个条件的所有可能结果至少出现一次，并且每个判定本身的所有可能结果也至少出现一次。可见判定/条件覆盖实际上是将判定覆盖和条件覆盖的基本思想组合，使测试的功能效果更强。

综合前面判定覆盖和条件覆盖的分析结果，发现最初满足条件覆盖的一组测试用例，同样满足判定/条件覆盖，如表 9-3 所示。

表 9-3 同时满足条件覆盖和判定覆盖的测试用例

测试用例	条件覆盖				判定覆盖	
	$A>1$	$B=0$	$A=2$	$X>1$	$A>1$ and $B=0$	$A=2$ or $X>1$
(1) $A=2,B=0,X=4$	真	真	真	真	真	真
(2) $A=1,B=1,X=1$	假	假	假	假	假	假
是否满足覆盖	满足				满足	

这里再另外给出一组测试用例，满足判定/条件覆盖，如表 9-4 所示。

表 9-4 满足判定/条件覆盖的测试用例

测试用例	条件覆盖				判定覆盖	
	$A>1$	$B=0$	$A=2$	$X>1$	$A>1$ and $B=0$	$A=2$ or $X>1$
(1) $A=3,B=0,X=0$	真	真	假	假	真	假
(2) $A=1,B=1,X=1$	假	假	假	真	假	假
(3) $A=2,B=1,X=3$	真	假	真	真	假	真
是否满足覆盖	满足				满足	

5）条件组合覆盖

条件组合覆盖就是选取足够的测试用例，使每个判定中条件结果的所有可能组合至少出现一次，它是比判定/条件覆盖更强的覆盖。

要分析满足条件组合覆盖的测试数据，首先要列出两个判定的各种条件组合情况，见表 9-5。

表 9-6 列出了所选测试用例覆盖上述条件组合的情况。显然，它们满足条件组合覆盖标准。

表 9-5　判定的各种条件组合情况

判　定	组　合	组合编号
$A>1$ and $B=0$	$A>1,B=0$	G1
	$A>1,B<>0$	G2
	$A\leqslant 1,B=0$	G3
	$A\leqslant 1,B<>0$	G4
$A=2$ or $X>1$	$A=2,X>1$	G5
	$A=2,X\leqslant 1$	G6
	$A<>2,X>1$	G7
	$A<>2,X\leqslant 1$	G8

表 9-6　满足条件组合覆盖的测试用例

测试数据	覆盖的条件组合
$A=2,B=1,X=2$	G2,G5
$A=0,B=0,X=0$	G3,G8
$A=4,B=0,X=3$	G1,G7
$A=0,B=1,X=3$	G4,G7
$A=2,B=0,X=0$	G1,G6

6）路径测试

路径覆盖是指选择足够的测试用例，使流程图中的每条路径至少经过一次。路径覆盖相当于判定组合覆盖，在上面的例子中，路径覆盖的测试用例如表 9-7 所示。

表 9-7　满足路径覆盖的测试用例

测试数据	覆盖的判定组合	所走的路径
$A=2,B=0,X=3$	$A>1$ and $B=0$ 为真，并且 $A=2$ or $X>1$ 为真	(a)(b)(c)(d)(e)(f)
$A=2,B=0,X=0$	$A>1$ and $B=0$ 为真，并且 $A=2$ or $X>1$ 为假	(a)(b)(c)(d)(f)
$A=0,B=0,X=3$	$A>1$ and $B=0$ 为假，并且 $A=2$ or $X>1$ 为真	(a)(b)(d)(e)(f)
$A=0,B=0,X=0$	$A>1$ and $B=0$ 为假，并且 $A=2$ or $X>1$ 为假	(a)(b)(d)(f)

路径覆盖是一种相当强的逻辑覆盖，但路径覆盖只需考虑每个判定表达式的取值，而并没有考虑在判定表达式内各种条件的组合，所以将路径覆盖和条件组合覆盖结合起来，就可以得到更强的覆盖。

2. 黑盒测试法

黑盒测试法与白盒测试法正好相反，它把程序看成一个不知内部结构的黑盒子，不管程序内部的结构与处理怎么样，从用户观点出发，按照程序的预定的功能和性能正常使用，检测程序是否能适当接受输入数据并产生正确的输出信息。因此，黑盒测试有两个显著优点：软件具体的实现与黑盒测试无关，因此如果软件具体的实现发生变化，测试用例仍然可用；从用户的角度出发，黑盒测试以软件规格说明书为依据选取测试数据，其正确性依赖于规格说明的正确性，输入数据不会因为实现的不同而不同，在一些情况下设计黑盒测试用例可以和软件实现同时进行，从而可以大大压缩项目总的开发时间。

黑盒测试有几种常用的方法，主要包括等价类划分、边界值分析、错误猜测和因果图等。

1）等价类划分

等价类划分是分步骤地把过多(无限)的测试案例减小到同样有效的小范围的过程，这是一种用例设计的思想。等价类划分是一种典型的黑盒测试法，也是一种非常实用的测试方法。使用这种方法进行程序测试时，首先要在分析需求规格说明书的基础上划分等价类，然后列出等价类表。

那么什么是等价类呢？等价类就是指某个数据域的集合，在这个集合里如果一个输入

条件作为测试数据不能发现程序中的错误，那么说明这个集合中的所有输入条件在测试时也不能发生错误(除非这个等价类的某个子集还属于另一个等价类)。

在寻找等价类时，想办法把软件的相似输入、输出、操作分成组。等价区间的划分没有一定的标准，只要足以覆盖测试对象就行了。人们在测试过程中，当一个测试用例发现了一个错误，往往就放弃检测，但是这个测试用例还有可能发现其他的错误，例如，客户在 ATM 上取款的单次取款金额在 50～1000 元，并且单次取款金额必须是 50 的整数倍，若一个测试用例的取款金额为 1002 元，在测试中很可能只测试出取款金额在 50～1000 元之外的错误，而忽视了单次取款金额必须是 50 的整数倍。对于测试非法输入的无效等价类来说就是如此，每个无效等价类都很有可能查出程序中的多个错误，为了避免某些错误被忽略，所以要为每一个无效等价类设计一个新的测试用例。所以，等价类又分为有效等价类和无效等价类。

有效等价类是指符合需求规格说明书要求的，有意义的，合理的输入数据所构成的集合，它主要用于测试程序是否实现了需求规格说明书中的功能要求。

无效等价类是指不符合需求规格说明要求的，非法的，无意义的输入数据所构成的集合，它主要用于测试程序是否做了需求规格说明书以外的事。

在划分等价类时，需要先研究需求，了解程序的功能，才能够确定输入数据的有效等价类和无效等价类。同时在确定输入数据的等价类时，常常还需要分析输出数据的等价类，从而根据输出数据的等价类导出输入数据的等价类。

划分等价类需要对系统功能的理解和划分等价类的经验，没有非常确定的方法，但是下面有几条启发式的原则可以供参考。

(1) 如果某个条件规定了输入值的范围，那么可以确定一个有效等价类和两个无效等价类。例如，客户在 ATM 机上取款的单次取款金额在 50～1000 元，因此有效等价类为 50≤取款金额≤1000；无效等价类就是取款金额小于 50 和取款金额大于 1000。

(2) 如果某个输入条件规定了值的个数，那么就可以确定一个有效等价类和两个无效等价类。例如，考生填报志愿必须从所有学校中选出 1～3 个，那么有效等价类为：1≤填报学校数目≤3；无效等价类为：填报学校数目<1 和填报学校数目>3。

(3) 如果某个输入条件规定了一个输入数据必须遵循的规则，例如，客户在 ATM 机上取款的单次取款金额必须是 50 的整数倍，那么，就可以确定一个满足此规则的作为有效等价类，不满足此规则的作为无效等价类。

(4) 如果规定了输入数据是整型，则可以划分出正整数、零和负整数 3 个有效类。

(5) 如果程序处理的对象是表格，则应该使用空表、含一项的表和含多项的表 3 个有效类。

以上列举的只是很小的一部分规则，实际应用时情况复杂多样，还应参照具体事例，灵活地划分等价类。正确划分等价类需要经验积累，正确分析所测程序的功能。另外要注意的是，那些对于编译程序能够暴露的错误，一般不设计测试用例。等价类划分完以后，就可以利用等价类来设计测试用例，其具体步骤如下。

(1) 设计一个新的测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类，重复这一步骤，直到所有的有效等价类都被完全覆盖为止。

(2) 设计一个新的测试用例，使其尽可能覆盖一个无效等价类，重复这一步骤，使所有

无效等价类都被完全覆盖。

可以将等价类划分测试用例设计整个过程用如图 9-4 所示的流程图来表示。

2）边界值分析

实践表明，设计一些边界测试条件用例，使程序运行在边界附近，这样常常会取得良好的测试效果。例如，数组的下标、条件循环的边界次数、条件判断的边界值等。边界测试条件是指相对于输入与输出等价类直接在其边界上或稍高于其边界，或稍低于其边界的状态条件。使用该方法设计测试用例，也需要经验和创造性，它需要先确定边界的情况，选取的测试数据要刚好等于、稍微小于和稍微大于边界值。如果系统的输入或输出是一个有序集（如线性表、表格等），那么选取测试用例就应注意有序集的第一个元素和最后一个元素。

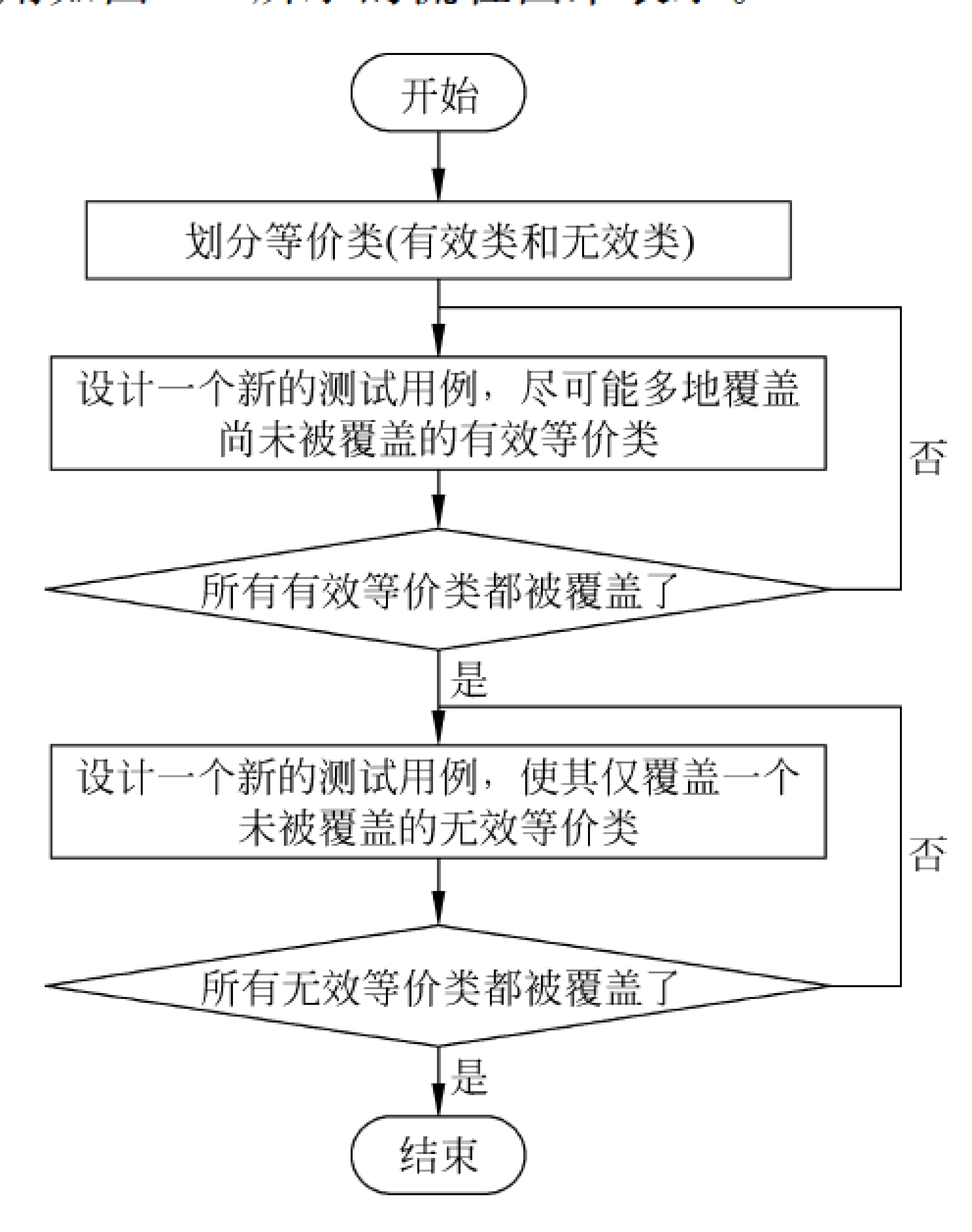


图 9-4　等价类划分测试用例设计流程

例如，在 ATM 系统中，客户在 ATM 机上取款时要求单次取款金额在 50～1000 元。其在 0、50、1000 这些边界点容易出错，极有可能因为编码疏忽使条件式“amount ≥ 50 and amount≤1000”漏掉等号“=”，利用边界值分析方法设计的测试用例为－1，0，1，49，50，51，999，1000，1001。

一般来说，在设计测试用例时，先使用等价类划分技术来获得测试用例，然后再使用边界值分析法对测试用例进行补充。

3）错误猜测

使用等价类划分和边界值分析可以帮测试人员设计出具有代表性的测试用例，但不同的程序又有一些特别的出错情况，这些特别情况不能被具有代表性的测试用例测试出来。一般来说，即使是较小的程序，可能的输入组合也很巨大，而具有丰富测试经验的测试人员往往能够直接找到程序特别容易出错的地方，因此说，人们的经验和直觉，在测试过程中也具有很大意义。错误猜测法主要是依靠测试人员的直觉和经验，即利用经验和直觉尽可能多地列出程序中可能出现的错误或容易发生错误的情况，然后根据它们开发测试用例。

4）因果图

等价类划分法和边界值分析方法都是着重考虑输入条件，但没有考虑输入条件的各种组合、输入条件之间的相互制约关系。这样虽然各种输入条件可能出错的情况已经测试到了，但多个输入条件组合起来可能出错的情况却被忽视了。对于稍大一些的系统，不可能穷尽所有的条件组合。如果在测试时必须考虑输入条件的各种组合，则可能的组合数目将是天文数字，因此必须考虑采用一种适合描述多种条件的组合，相应产生多个动作的形式来进行测试用例的设计，这就需要利用因果图（逻辑模型）。

（1）因果图的基本图形符号

因果图是一种利用图解法分析输入的各种组合情况，从而设计测试用例的方法，它适合

检查程序输入条件的各种组合情况。这里先介绍因果图的基本图形符号。

在图 9-5 中，基本图形符号左边结点表示原因状态，右边结点表示结果状态，原因状态或结果状态的可取值为“0”和“1”，其中“0”表示某状态不出现，“1”表示某状态出现。在图 9-5 中，有 4 种因果关系，依次是恒等、非、或、与，它们的具体含义如下。

- 恒等：表示原因与结果之间是一一对应的关系，例如，如果 $A=1$，那么 $C=1$；如果 $A=0$，那么 $C=0$。
- 非：表示原因与结果之间是相反的关系，例如，如果 $A=1$，那么 $C=0$；如果 $A=0$，那么 $C=1$。
- 或：表示几个原因中任一个结果出现时结果都出现，例如，如果 $A=1$ 或 $B=1$，那么 $C=1$；如果 $A=B=0$，那么 $C=0$。
- 与：表示所有原因都出现时结果才出现，例如，如果 $A=B=1$，那么 $C=1$；如果 $A=0$ 或 $B=0$，那么 $C=0$。

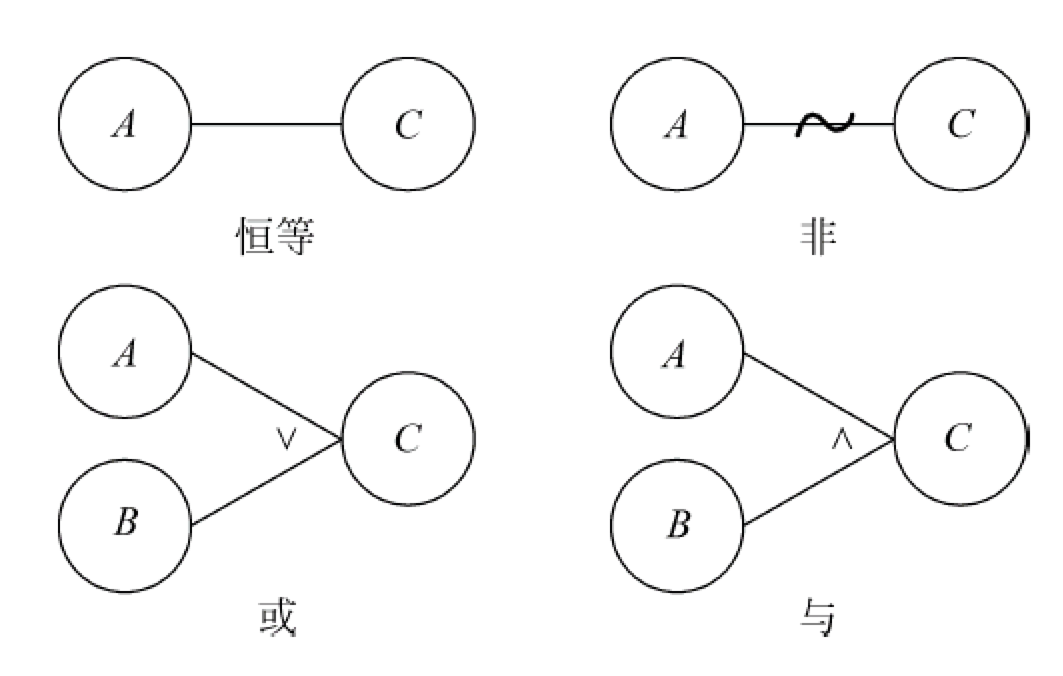


图 9-5　因果图的约束条件符号

（2）因果图的约束条件符号

约束条件符号如图 9-6 所示，其含义如下。

- 互斥：为原因与原因之间的关系，表示 A、B 两个原因不同时成立，最多只有一个可能成立，即 A、B 中至多只有一个 1。互斥用 E 表示。
- 包含：为原因与原因之间的关系，表示 A、B、C 这 3 个原因中至少有一个必须成立，即 A、B、C 中不可全部为 0。包含用 I 表示。
- 唯一：为原因与原因之间的关系，表示 A、B 中必须只有一个成立，即 A、B 中仅有且只有一个 1。唯一用 O 表示。
- 要求：为原因与原因之间的关系，表示当 A 出现时，B 也必须出现，即如果 $A=1$，B 必须为 1。要求用 R 表示。
- 屏蔽：这个约束条件最为特殊，不同于以上是对输入原因的约束，而是从输出结果考虑。当 A 为 1 时，B 必须是 0，反之，当 A 为 0 时，B 值不确定。屏蔽用 M 表示。

采用因果图设计测试用例的具体步骤如下。

- 根据需求规格说明，分析哪些是“原因”（输入条件的等价类），哪些是“结果”（输出条件或程序状态的修改），并要给每个原因或结果赋予一个编号。
- 根据需求规格说明，找出原因与原因之间，原因与结果之间对应的关系，根据这些关系画出因果图。

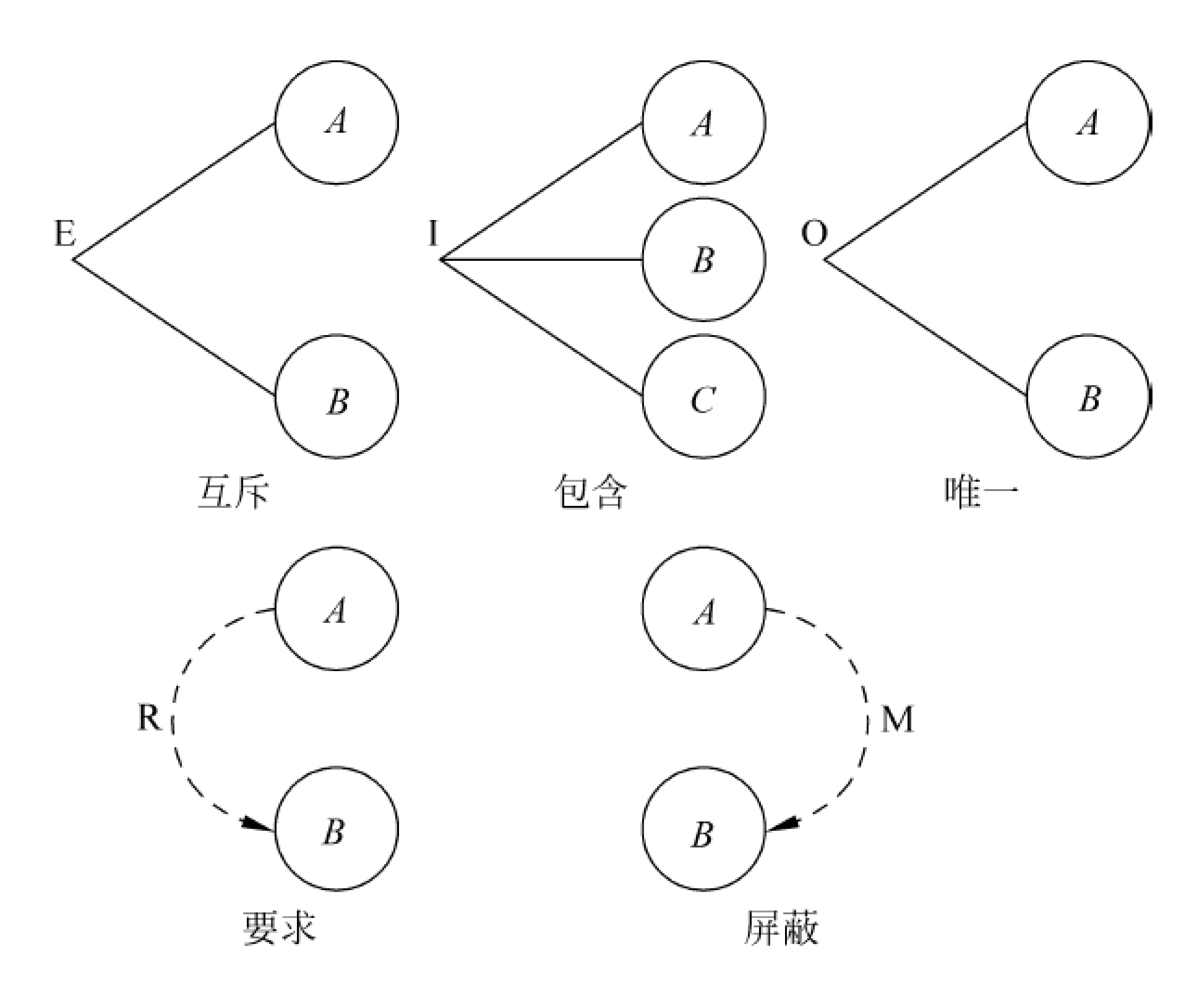


图 9-6 因果图的约束条件符号

- 在某些情况下，有些原因与原因之间、原因与结果之间的一些组合情况不可能出现，由此，因果图上就会用一些标号标明约束或限制条件来表明其含义。
- 将得到的因果图转换成判定表。
- 用判定表中每一列作为依据来设计测试用例。

以下对因果图进行举例说明。

在 ATM 系统的例子中，使用者进行取现操作或进行转账操作时(当然是不能同时又取现又转账)需要输入操作金额。在进行转账操作时，输入的操作金额必须是合法的正数，即满足一个正则表达式，定义为条件 1，如果满足条件 1，则进入输入对方账号界面，如果不满足，则提示“操作金额有误”；而在取现操作时，输入的操作金额就有更多的要求，因为 ATM 机只能提取 50 元和 100 元两种面额的人民币，另外，每一次操作的取款金额不能大于 1000 元，所以要求使用者输入的金额必须是 50 的倍数，且大于等于 50，小于等于 1000，把这个条件定义为条件 2，如果输入的金额同时满足条件 1 和条件 2，则进入取款程序，如果不满足条件 1，则提示“操作金额有误”，如果不满足条件 2，则提示“输入的金额必须是 50 的倍数，且大于等于 50，小于等于 1000”。

第一步分析原因和结果，并给予编号。

列出原因并编号：

1：输入金额满足一个正则表达式

2：输入的金额是 50 的倍数，且大于等于 50，小于等于 1000

3：取现操作

4：转账操作

列出结果并编号：

21：进入输入对方账号界面

22：提示“操作金额有误”

23：进入取款程序

24：提示“输入的金额必须是50的倍数，且大于等于50，小于等于1000”

第二步是找出原因与原因之间、原因与结果之间对应的关系，画出因果图，并用一些标号标明约束或限制条件来表明其含义，如图9-7所示(其中11和12是中间结果)。

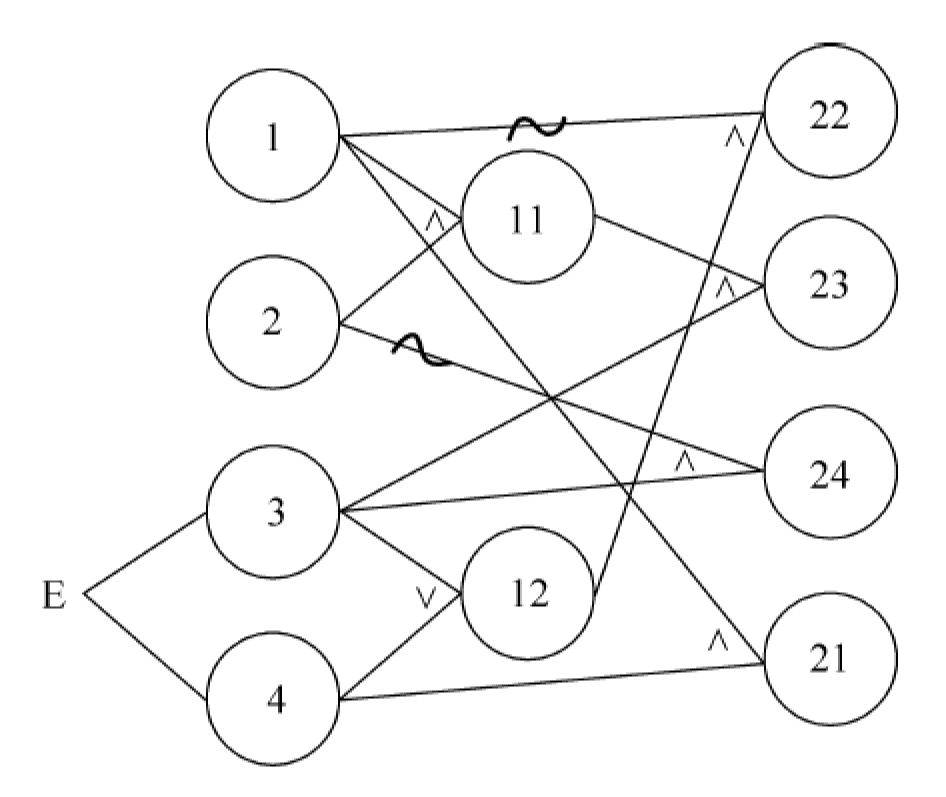


图9-7 ATM系统取款金额因果图实例

3. 黑盒测试与白盒测试的优缺点总结

从测试的依据来看，黑盒测试是根据程序内部结构进行结构测试，它能对程序内部的指定部位进行覆盖测试。它的缺点是无法检测程序的外部特性，无法对不符合或不满足规格说明的程序部分进行测试。它的主要方法是逻辑覆盖技术，具体包括语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、路径覆盖等。

白盒测试的测试依据是根据需求规格说明书进行功能的测试，它能够从用户的角度出发，站在用户的立场上进行测试。它的缺点是不能测试程序内部指定部位的错误。它的主要方法有等价类划分、边界值分析、错误推测法、因果图等。

黑盒测试和白盒测试各有优缺点，它们在测试实践中都非常有效而且都很实用。实践中要将它们相互结合，取长补短，不应有所偏废。

一般来说，在单元测试时多数采用白盒测试，而在确认测试或系统测试中多数采用黑盒测试。这里提到了单元测试、确认测试、系统测试，这些都是测试步骤中的概念，下面将继续介绍软件测试步骤。

9.3.3 软件测试步骤

软件测试同软件开发类似，也需要分阶段来进行，且与软件开发的阶段存在对应关系，软件开发的阶段是需求分析→概要设计→详细设计，而软件测试的阶段则刚好与此相反，整个软件测试工作可以分为4个步骤：单元测试(与详细设计对应)，集成测试(与概要设计对应)、确认测试和系统测试(与需求分析对应)。

图9-8描绘了软件测试工作的4个步骤，第一步是单元测试，主要集中在对每个单元内部源代码进行测试，以确保各个单元模块能正常实现功能。本次测试大量采用了白盒测试法，主要是尽可能地发现模块内部的代码错误。第二步是集成测试，即把已经通过单元测试的模块组合起来，针对设计信息的要求进行集成测试，其测试目的主要是检验与软件设计相关的体系结构的构造问题。黑盒测试法在该阶段用得比较多。通过集成测试后，要对已实现的软件是否满足开发工作初期指定的确认准则和软件配置是否完全正确进行检测。这就是第三步的确认测试，即检测所开发软件能否满足所设计的功能和性能的最后测试，本阶段也通常采用黑盒测试法。通过本阶段测试后，就进入第四步的系统测试，即必须把软件放入实际运行环境与系统中的其他部分进行协调测试。严格意义上来说，系统测试已超出了软件工程的范畴。

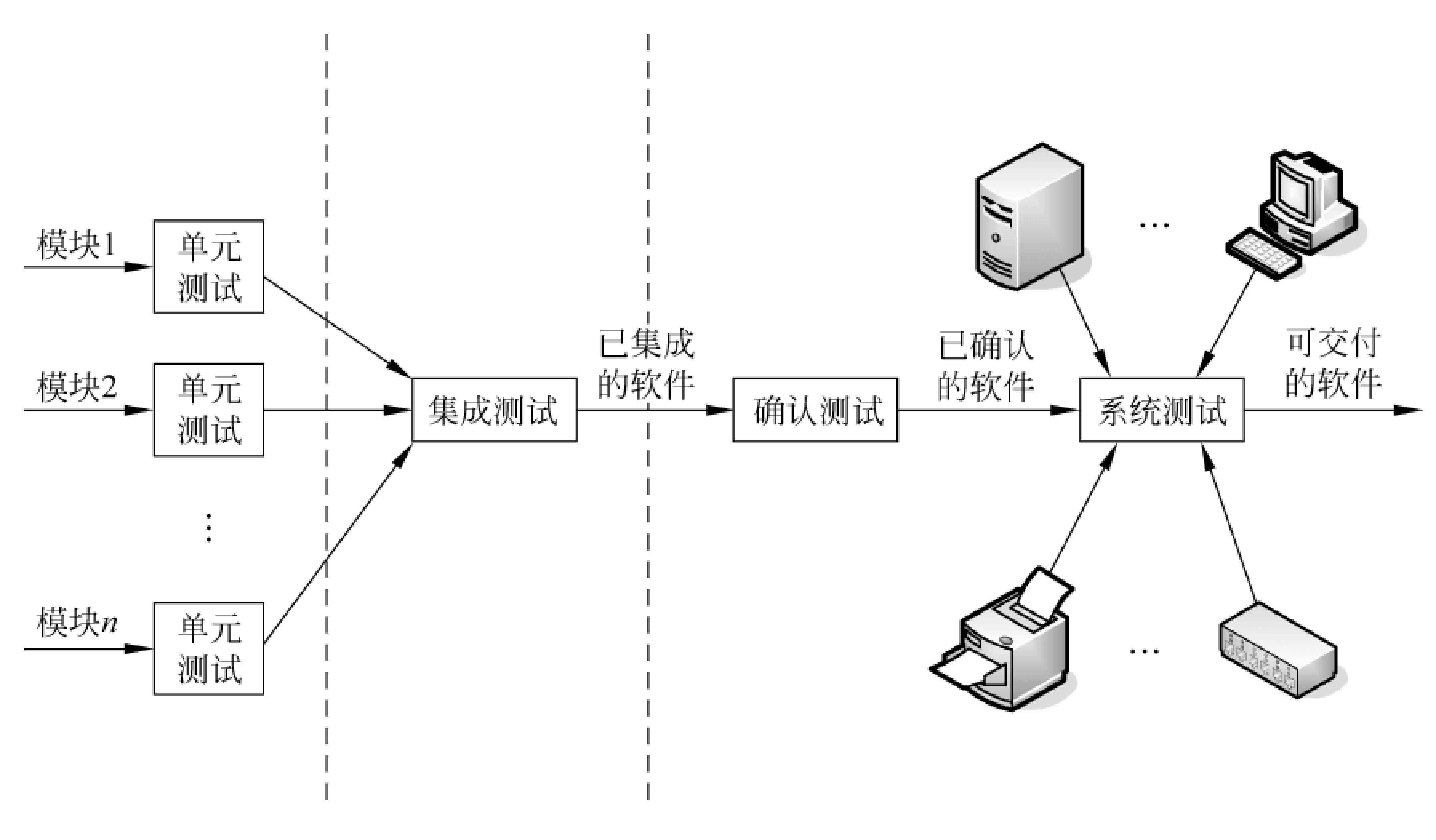


图 9-8 软件测试步骤

以下对软件测试的 4 个步骤进行详细的解释。

1. 单元测试

单元测试，是针对软件中最小单元（模块）内部进行的测试，所以又称为模块测试。一般来说，在完成编码之后，紧接着就要进行单元测试。单元测试的责任人主要是编码人员，一般采用白盒测试法，从程序的内部结构出发设计测试用例，单元测试的目的是发现各模块内部可能存在的各种错误。如果不同的模块是并行开发的，那么多个程序模块单元可以并行地独立开展测试工作。

1）单元测试的内容

单元模块是单元测试的测试对象，它是组成系统的最小模块单位，如函数、过程等。在单元测试之前，测试员要仔细阅读详细设计说明书和源程序，对模块的内部结构有较全面的理解。

2）单元测试步骤

模块不是完全独立的，需要考虑其与外界的联系，所以需要设置若干个辅助测试模块，辅助测试模块分为两种，一种是驱动模块（Driver），另一种是桩模块（Stub），如图 9-9 所示。

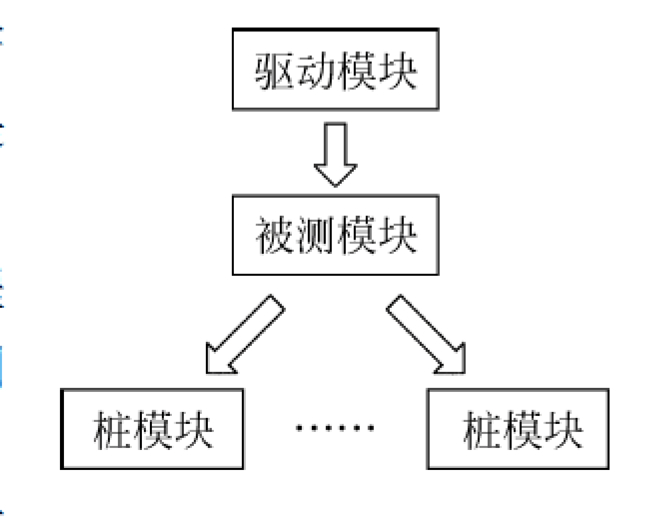


图 9-9 驱动模块和桩模块

驱动模块是模拟被测模块的上级，相当于被测模块的主程序，通过驱动模块调用被测模块，接收测试数据，然后把这些测试数据传送给被测模块。

桩模块是作为被测模块所调用的子模块形式出现的，桩模块受被测模块的调用，它可以检验调用参数，并且模拟被调用子程序模块的功能，最后把结果传送给被测模块。如果被测模块是顶层模块，测试时不需要驱动模块，同样的道理，被测模块是底层模块，测试时则不需要桩模块。

在进行单元测试时，被测模块和驱动模块、桩模块共同构成一个“单元测试环境”。

2. 集成测试

单元测试是对模块的测试,但模块连接起来是否工作正常呢? 为了保证模块连接起来能够正常工作,需要按照设计时做出的结构图,把各模块组装起来,进行集成测试。集成测试的任务包括对系统进行组装和检测。首先,集成测试需要将经过测试的模块组织起来,发现模块是否能按结构设计组装,接着在系统装配的过程中,又要对每个集成到系统中的模块进一步检测,发现诸多模块之间是否具有良好的协作性。此次测试通常采用黑盒测试法完成。集成测试主要是检查模块间的接口通信,发现设计阶段产生的问题。

以下列举了一些集成测试涉及的具体测试任务。

- 在将模块连接起来时,测试模块间接口的数据是否会丢失。
- 测试一个模块的功能对另一个模块的功能是否会产生不良的影响。
- 将子功能组合时,测试系统能否达到设计预期的协同效果。
- 测试全局数据结构是否有问题。
- 测试计算误差是否会从一个模块传递到其他模块,而被放大到不可接受的地步。

通常的组织集成测试有两种方式:非渐增式测试和渐增式测试。

1) 非渐增式集成测试

它是在配备辅助模块的条件下,先测试所有的单元模块,在此基础上,按照程序结构图同时把所有模块放在一起,对程序进行集体测试。如果模块之间存在问题,也只有在最后集成后才能暴露出现。测试面对的场面常常是混乱不堪的,需要同时面对各种错误,定位和修改这些错误都是非常困难的事情,有时修改了一个错误,又有新的错误出现,循环往复看似没有尽头,非常打击开发人员的士气。如图 9-10 所示,采用非渐增式测试方法来测试时,先分别单独测试模块 1、模块 2、模块 3、模块 4、模块 5、模块 6、模块 7、模块 8,然后再连接起来进行整体测试。

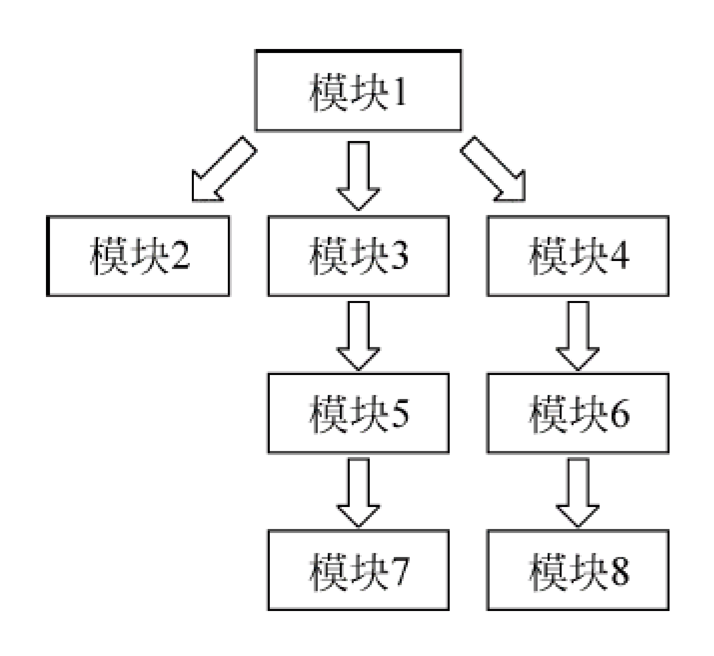


图 9-10　非渐增式集成测试例子

2) 渐增式集成测试方法

它与非渐增式测试方法有所不同,它把程序划分成小段来构造和测试,即把单元测试和集成测试合并在一起。根据模块结构图,按某一次序选一个还未测试的模块,把它同已测试好的模块合并在一起进行测试,每次增加一个单元模块,直到所有的都集成在一起成为一个完整的程序。根据模块测试次序的不同,渐增式集成测试又可分为自顶向下集成和自底向上集成。下面讨论这两种渐增式集成策略。

(1) 自顶向下集成

该策略是从主控制模块开始,沿着软件的控制层次逐步向下将各个模块组合起来。这里既可以使用广度优先策略,也可以使用深度优先策略。在上面的例子中,使用广度优先策略进行集成(图 9-11),模块的测试集成顺序是模块 1→模块 2→模块 3→模块 4→模块 5→模块 6→模块 7→模块 8;如果使用深度优先策略进行集成(图 9-12),模块的测试集成顺序是模块 1→模块 2→模块 3→模块 5→模块 7→模块 4→模块 6→模块 8。

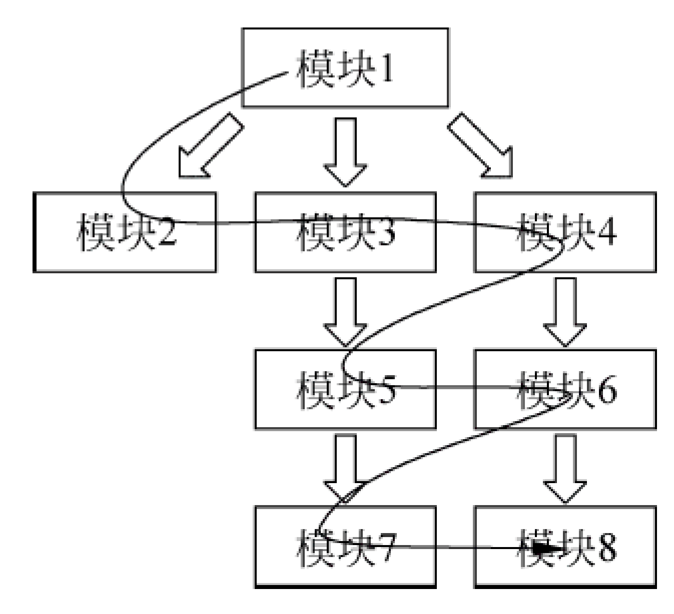


图 9-11　广度优先自顶向下集成

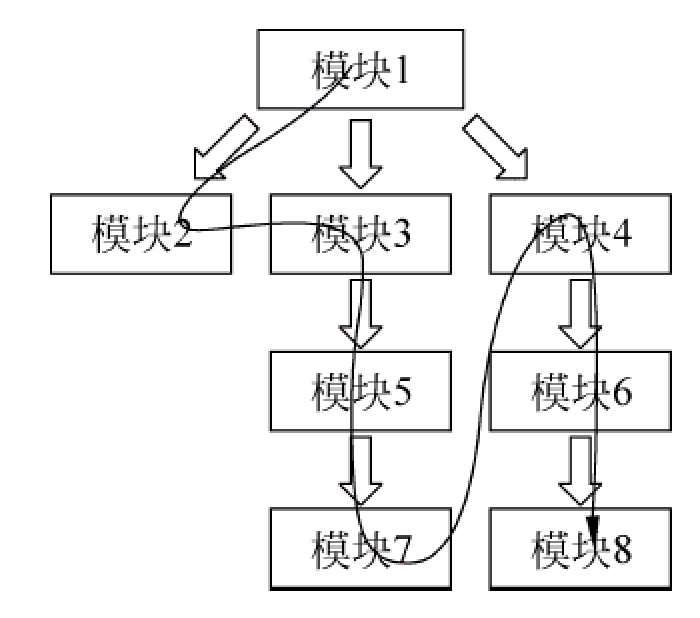


图 9-12　深度优先自顶向下渐集成

以下是自顶向下渐增式集成测试的具体步骤。

第一步：对主控制模块进行测试，对于所有主控制模块的子模块用桩模块来代替。在例子中，先测试主控制模块 1，用对应的桩模块来代替模块 2、模块 3、模块 4。

第二步：根据优先策略（广度优先或深度优先）来确定每次使用哪个真正的模块来代替桩模块，当然这个真正的模块可能需要新的桩模块。在例子中，如采用深度优先策略进行，当对模块 3 测试集成完成之后，用模块 5 替换掉对应模块 3 的桩模块，同时引入代替模块 7 的桩模块，作为模块 5 的子模块。然后对模块 5 进行测试。

第三步：可能需要进行部分或全部的回归测试（即重复以前做过的测试），以保证加入模块后没有引进新的错误。

第二步、第三步是重复进行的过程，根据集成策略（广度优先策略、深度优先策略）逐步构造起完整的软件结构。自顶向下的集成策略能够在测试的早期对主要的控制或关键的抉择进行检验。在一个合理的软件结构中，主要的控制或关键的抉择往往位于结构的较上层，如果它们出现问题，自顶向下的集成策略可以有助于较早发现这类问题，并及早找出对策。另外，在项目开发初期，如果能够实现一个完整的功能，能够增强开发人员和用户的信心，结合使用深度优先策略，可以在早期实现并验证一个完整的功能。

当然，自顶向下说起来挺简单，实际操作还是有一些问题值得注意的。最常见的问题是：由于使用桩模块来代替子模块进行测试，在测试软件系统的较高层次时，需要较低层次上的处理，而较低层次上是使用桩模块来代替的，它无法进行某些处理，这样有些测试就无法进行。面对这种情况，有两种方法：测试人员只能把许多测试推迟到用真实模块代替桩模块后再进行；或者从结构的底部往上集成测试。于是就提出了另一种渐增式集成测试的方法：自底向上集成的测试方法。

（2）自底向上集成

它是从软件结构的最底层模块开始测试集成的。自顶向下需要桩模块而不需要驱动模块，与此对比，自底向上则是需要驱动模块而不需要桩模块，以下是自底向上渐增式集成测试的具体步骤。

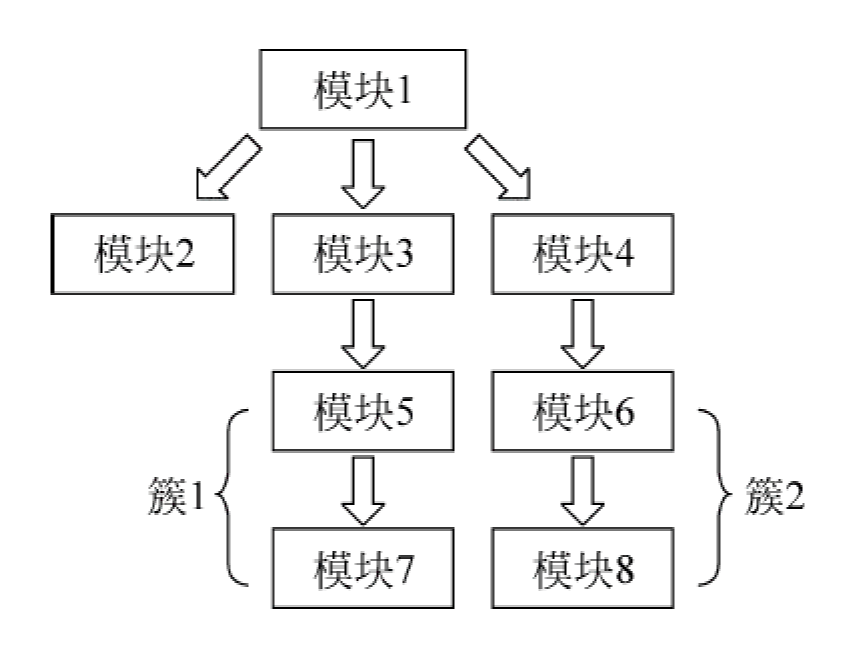


图 9-13　自底向上集成

第一步：把底层的模块整合成具有一定功能的簇，如图 9-13 所示的簇 1 由底层模块 7 和模块 5 组成，簇 2 由底层模块 6 和模块 8 组成。

第二步：用驱动模块来驱动相应的簇，控制测试数据的输入和输出，如图中代替模块 3 的驱动模块控

制簇 1。

第三步：利用测试用例来测试子功能簇，如图中簇 1。

第四步：用对应模块替代驱动模块，如图中用模块 3 替代驱动模块，这样就向上集成形成更大的子功能簇。

重复循环执行第二步到第四步，就可以不断向上集成，最终形成完整的结构和功能。

3. 确认测试

在集成测试过程中，各个单元模块被连接在一起，测试人员将各模块之间的接口参数等种种问题测试出来并交由开发人员解决，之后，测试工作进入了确认测试阶段。确认测试的目的就是验证该测试软件的功能和性能及其他特性能否达到用户的要求，如果能够达到这些功能性能要求，那么就认为软件是合格的，所以有时把这种测试称为合格性测试。软件需求规格说明是软件有效性的标准，是进行确认测试的基础，它准确地描述了用户对软件的合理期望。

1）进行有效性测试

到底什么是软件的有效性？软件的有效性是指软件的功能和性能满足用户的合理期待。在需求规格说明书中制订测试计划，规定要做测试的种类，在模拟的环境中（或开发环境）运用黑盒测试的方法，验证测试软件能否满足比测试计划更详细更具体的测试规格说明书，从而确定软件的功能和性能能否与客户需求相符。除了考虑软件功能、软件性能和所有文档都是正确的以外，还需检验其他的软件需求。

确认测试无非两种结果：软件是有效的，即功能和性能符合用户的要求；或者软件的功能和性能与用户的要求存在偏差。第一种结果当然皆大欢喜，第二种结果是确认测试的目的。在确认测试时所发现的第二种结果常常和需求分析阶段的差错有关系，涉及的面较广，所以有时需要和用户进行协商。

2）配置复查

配置复查是确认测试的一个重要内容，它的目的在于确保已开发软件的所有成分都已经齐全，各成分的质量都符合要求，文档与程序保持一致并进行了分类编目，完全能够保证投入运行以后相应软件的维护工作，因此也称之为配置审计。

大型通用软件，在正式发布前，通常需要执行 Alpha 和 Beta 测试，目的是从实际终端用户的使用角度，对软件的功能和性能进行测试，以发现可能只有最终用户才能发现的错误。

Alpha 测试由用户在开发者的场所进行，并且在开发者对用户的“指导”下进行测试。开发者负责记录发现错误和在使用中遇到的问题。总之，Alpha 测试是在受控的环境中进行的，Alpha 测试不能由程序员或测试员完成。Alpha 测试发现的错误，可以在测试现场立刻反馈给开发人员，由开发人员及时分析和处理。其目的是评价软件产品的功能、可使用性、可靠性、性能和支持，尤其注重产品的界面和特色。Alpha 测试可以从软件产品编码结束之后开始，或在模块（子系统）测试完成后开始，也可以在确认测试过程中产品达到一定的稳定和可靠程度之后再开始。有关的手册（草稿）等应该在 Alpha 测试前准备好。

Beta 测试也不能由程序员或测试员完成，它是由软件的最终用户们在一个或多个客房场所进行，Beta 测试是软件在开发者不能控制的环境中的“真实”应用。用户在 Beta 测试过程中遇到的一切问题（真实的或想象的），应定期地报告给开发者。接收到在 Beta 测试期间

报告的问题之后,开发者对软件产品进行必要的修改,并准备向全体客户发布最终的软件产品。Beta测试着重于产品的支持性,包括文档、客户培训和支持产品的生产能力。只有当Alpha测试达到一定的可靠程度后,才能开始Beta测试。由于Beta测试的主要目标是测试可支持性,所以Beta测试应该尽可能由主持产品发行的人员来管理。

由于Alpha和Beta测试的组织难度大、费用高、随机性强、周期跨度长以及测试质量效率难于保证,目前,专业测试服务机构大量涌现,很多软件的Beta测试由开发商外包给这些专业测试机构进行。

4. 系统测试

系统测试已完全超出了软件工程的范围,它是在分别完成集成测试和确认测试之后,确保各组成部分不仅已通过了单独的检验而且在系统各部分协调工作的环境下也可正常工作,然后再把软件、硬件和环境连在一起进行全面的测试。换个角度,系统测试实质是针对系统中各个组成模块进行的综合性检验,测试某个具体系统时可根据具体情况选择其中的几种:功能测试、恢复测试、紧张度测试、安全性测试、使用性测试、性能测试、容量测试、可靠性测试、文档测试、工序测试等。

9.4 调试

测试的任务是发现错误,而调试的任务是诊断和改正程序中的错误。调试是发现程序错误之后的排错过程。开发人员得到测试结果后,需要将错误的症状与错误的原因联系起来,这是一项技巧性的工作,有些错误表象与内在原因看上去没有任何联系,这就需要开发人员的经验来将二者联系起来。

测试与调试紧密关联,图9-14表示测试和调试的交互过程。

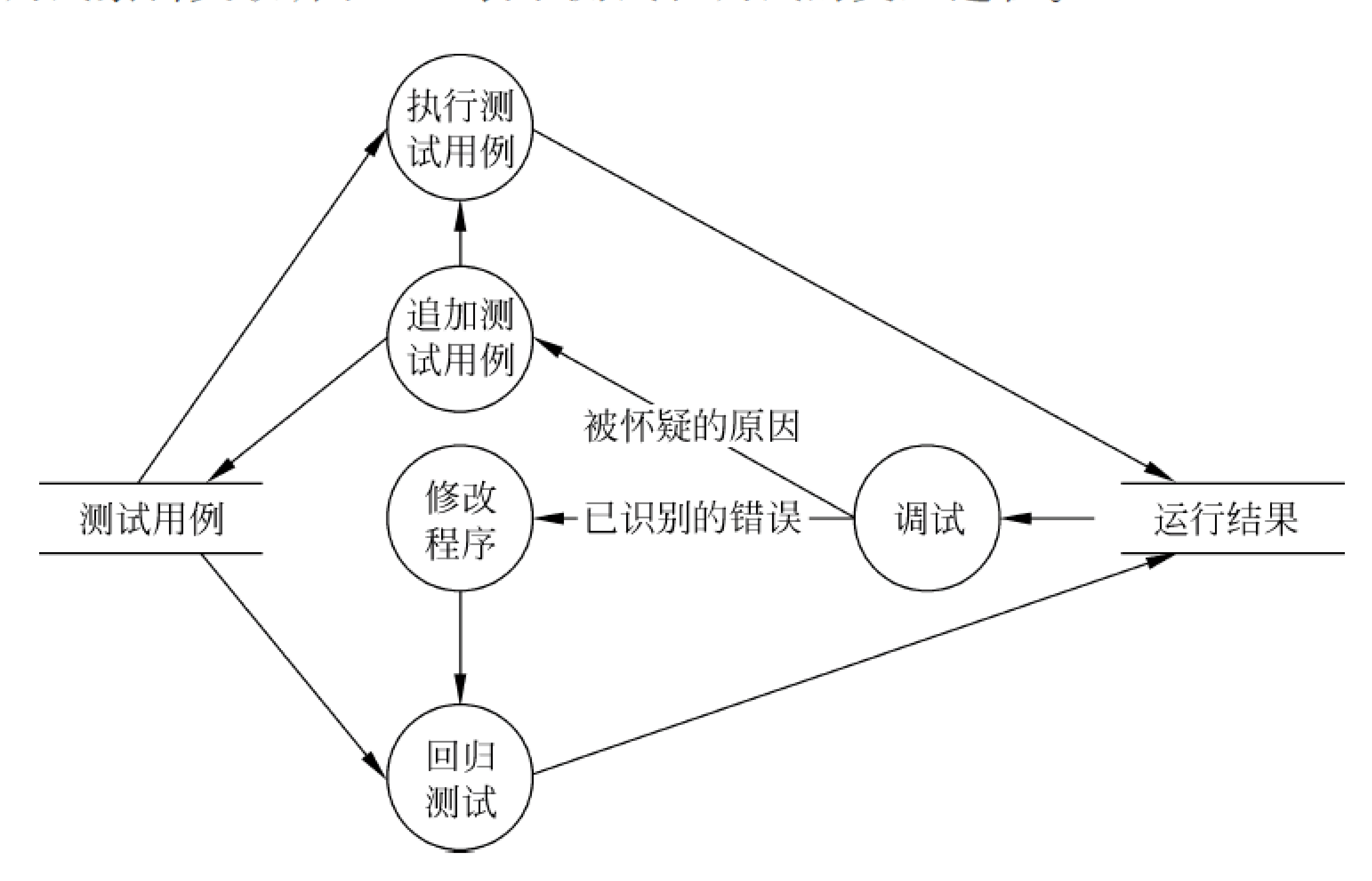


图9-14　测试和调试的交互过程

调试的过程无非有两种结果,一是识别出错误的原因并将程序修改正确;二是无法识别错误的原因,那么开发人员需要根据经验怀疑一个原因,然后设计相应的测试用例,验证

这个原因,循环往复直到找出原因并且修改正确。从图 9-14 中可以看出调试的过程。有时调试之所以困难,是因为错误的种类千奇百怪,调试的过程是对开发人员的巨大的脑力挑战,有时,错误会带来严重的后果,由此导致巨大的压力,调试过程中就会因为烦躁导致更多的错误。

1. 调试技术

较为常用的调试技术主要是使用调试工具。使用调试工具,主要利用程序设计语言的调试功能或使用专门的软件工具分析程序的动态行为。可供利用的典型程序设计语言调试功能有输出有关语句执行、子程序调用和更改指定变量值。用于调试的软件工具的共同功能是设置断点,即当执行到特定的语句或改变特定变量的值时,程序突然停止执行,程序员可以在终端上观察程序此时的具体状态。

2. 调试策略

调试的根本目的是找出软件错误的原因,并且将软件错误加以修改。虽然软件错误的原因千奇百怪,但也有一些常用的调试策略。

1) 试探法

试探法可能是最常用的一种调试策略,如果开发人员有丰富的调试经验,也不失为一种有效的方法,但如果是一个新手,更多的情况下只会浪费时间和精力。在这种调试策略下,调试人员分析错误征兆,猜想出故障的大致位置,然后使用前面的一些调试技术,如在程序中故障的大致位置处写上打印语句,获取程序中被怀疑的地方附近的信息,希望在这些地方发现错误原因的线索。

2) 回溯法

回溯法在调试小程序时通常非常有效,它是一种常用的调试方法。调试人员检查错误征兆,首先确定最先发现“症状”的地方,然后人工沿着程序的控制流往回追踪源程序代码,最终找出错误的根源或确定故障的范围为止。回溯法的另一种形式是正向追踪,就是使用输出语句检查一系列中间结果,从而确定最先出现错误的地方。回溯法对于小程序而言,往往能把故障范围缩小为程序中的一小段代码,仔细分析这段代码就不难确定故障的准确位置。相反,随着程序规模扩大,应该回溯的路径数目也会变得越来越大,以致彻底回溯变成不可能。

3) 归纳法

通常认为经过周密的思考可以找出大多数故障,归纳法就是这么一种系统化的思考方法。归纳法就是从个别推断一般的方法,这种方法从错误征兆出发,通过分析这些错误征兆之间的关系而找出故障。它主要有下述 4 个步骤。

(1) 收集有关的数据。收集程序执行得到的数据(可能正确也可能是错误的)。

(2) 组织整理数据。分析、比较、整理所收集的数据,通过观察数据间的关系,判断何种条件导致错误,何种条件不发生错误,由此得到一些规律。

(3) 导出假设。从错误的规律出发进行研究和推测,试着提出导致错误的一个或多个假设。如果无法做出推测,就需要设计更多的测试用例,得到更多的数据证据,来帮助导出假设。如果导出多个假设,通常选择可能性最大的那个假设。

(4) 对假设进行证明。可以使用收集来的数据进行验证假设,或者通过检查程序找到出错的原因。如果没有经过验证,就根据假设改正错误,通常情况下只能消除错误的症状或者改正部分的错误。如果无法验证假设,那就需要提出新的假设,进行新的验证。

4) 演绎法

演绎法从一般原理或一般前提出发,经过删除和细化的过程推导出结论。用演绎法调试,首先列出所有可能成立的原因或假设,然后一个个排除列举出的原因,最后,证明剩下的原因确实是错误的根源。演绎法主要分为以下 4 个步骤。

(1) 设想可能的原因。根据错误的症状,假设所有可能产生错误的原因,并以假设的形式将其列举出来。

(2) 用已有的数据排除不正确的假设。根据已有的数据仔细分析,寻找违反上一步假设的证据,尽可能排除上一步列出的假设。如果所剩假设多于一个,则选择可能性最大的那个;如果所有假设都被排除,则需要提出新的假设。

(3) 具体化余下的假设。对选定的假设进行进一步具体化,便于精确确定错误的位置。

(4) 证明假设。用收集来的数据对具体化的假设进行验证。这个过程和归纳法的第(4)步是相同的。

9.5 软件测试文档

1. 测试文档

测试文档是对将要进行的软件测试及测试结果的描述。因为软件测试与开发中的其他阶段的工作是有联系的,它实际上是一个复杂的过程,所以只有把对测试的要求、过程及其结果以正式的文件形式写出来,才能保证软件的质量和软件的正常运行。测试文档的出现是测试工作规范化的一个重要组成部分。

测试文档实际上贯穿于整个软件开发的过程中。在开发的需求分析阶段,就需要开始着手制定测试文档。为了验证设计阶段的一些设计方案,在测试文档中要能够体现这样的验证。测试文档对于测试阶段工作的指导与评价作用更是显著。最后,在软件投入运行的维护阶段,通常还要进行再测试或回归测试,这时还会用到测试文档。

测试文档的编制必须保证一定的质量和规范。只有这样才能使测试文档更有助于程序员编制程序,有助于管理人员监督和管理软件的开发,一个成功的测试文档在描述上要清晰准确,文档的结构和表述规范,同时考虑到工作的渐进性,要注意对文档进行必要的版本控制。

2. 测试文档的类型

根据测试文件所起的作用不同测试文档分为两类:测试计划和测试分析报告。

1) 测试计划

详细规定了测试的要求和内容,它的编写一般从需求分析阶段开始,到软件设计阶段结束时完成,涉及软件的需求和软件的设计。主要包括主测试计划、验证测试计划和确认测试计划。

2）测试分析报告

经过测试证实了软件具有的功能以及它的缺陷和限制，并给出评价的结论性意见。测试分析报告是对软件质量的评价，可以作为决定该软件能否交付用户使用的一个依据。测试分析报告在测试阶段内编写，以反映测试工作的情况。

本章小结

编码的任务就是把软件设计转换成计算机可以接受的程序代码。程序设计风格就是人们在长期的编程实践中形成的一套独特的习惯做法和编程方式。程序设计语言可以分为低级语言和高级语言两大类。

测试的目的是以最少的时间和人力找出软件中潜在的各种错误和缺陷。测试用例通常指对一项特定的软件产品进行测试任务的描述，体现测试方案、方法、技术和策略。测试用例是设计和制定测试过程的基础。动态测试是软件测试的主要方法，可分为两类：白盒测试法和黑盒测试法。

白盒测试法是把程序装在一个透明的白盒子里，即完全了解程序的结构和处理过程，按照程序内部的逻辑过程，来检验程序的每条通路是否都能按照预定的要求正确工作。逻辑覆盖是设计白盒测试方案的一种技术。根据覆盖的程度由低到高有语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、路径测试。

黑盒测试法把程序看成一个不知内部结构的黑盒子，不管程序内部的结构与处理怎么样，从用户观点出发，按照程序的预定的功能和性能正常使用，检测程序是否能适当接受输入数据并产生正确的输出信息。黑盒测试有几种常用的方法，主要包括等价类划分、边界值分析、错误猜测和因果图等。

软件测试步骤可以分为 4 个，即单元测试（与详细设计对应）、集成测试（与概要设计对应）、确定测试和系统测试（与需求分析对应）。

调试是发现程序错误之后的排错过程，其任务是诊断和改正程序中的错误。

思考与练习

1. 在进行软件开发时，如何选择程序设计语言？

2. 什么是程序设计的风格？为了具有良好的程序设计风格，应注意哪些问题？

3. 为什么说软件测试是软件开发中不可缺少的重要一环，但不是软件质量保证的“安全网”？

4. 为什么说穷举测试是不可能的？

5. 程序规格说明如下：要求输入 1800—2006 年中的某个年份，判断该年份是不是闰年。闰年的条件是能被 4 整除但不能被 100 整除或能被 100 整除且能被 400 整除。

（1）请根据上述的规格说明设计测试用例。

（2）程序的流程图如图 9-15 所示，请找出错误。

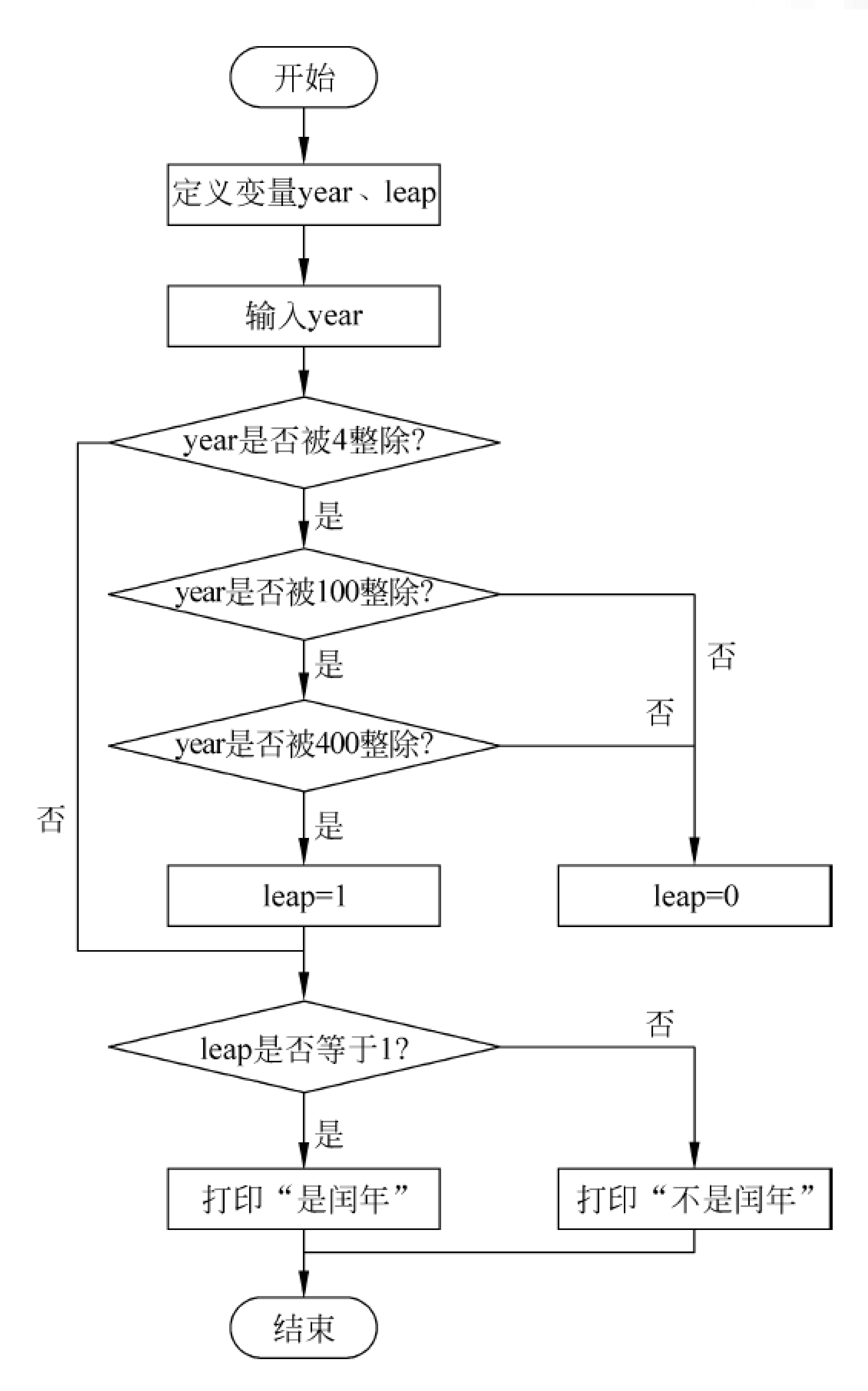


图 9-15　判断闰年的程序流程图

6. 当黑盒法测试认为程序运行正常时，白盒法测试却可能发现一个错误。请举例说明。

7. 一个关于判别三角形种类的程序：输入 3 个整数，作为三角形的 3 条边，程序根据输入值，判定分析后输出这 3 条边可以组成的是一般三角形、等腰三角形、等边三角形，还是非三角形。请根据这个程序的功能要求编写测试用例。

8. 如图 9-16 所示是一个程序流程图，现提供以下测试用例，试在其中选择最少的测试用例分别实现语句覆盖、条件覆盖、条件组合覆盖和路径覆盖。

用例 1：$A=0,B=0,C=0$；

用例 2：$A=0,B=1,C=1$；

用例 3：$A=1,B=0,C=0$；

用例 4：$A=1,B=1,C=0$；

用例 5：$A=1,B=1,C=1$；

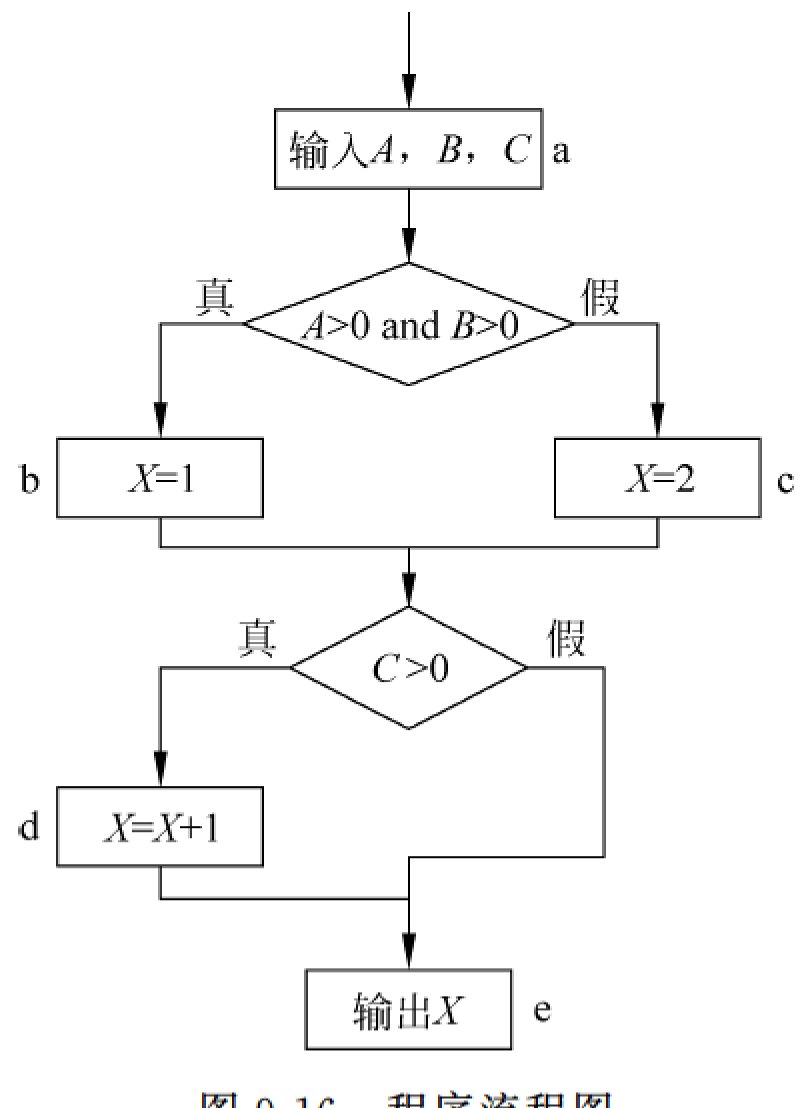


图 9-16　程序流程图



第10章 软件的技术度量

不能度量，就无法控制。

——Tom DeMarco（美国著名软件科学家）

10.1 软件度量的基本概念

10.1.1 测量、测度、度量、指标和估算

从本质上讲，工程是量化的学科。软件工程项目的定量描述涉及测量、测度、度量、指标、估算等概念①。

（1）测量

对产品或过程的某个属性的范围、数量、维度、容量或大小提供定量的指标，例如，对一个程序模块所含代码行数、符号数、文档页数等的测量。

（2）测度

测量的一个行为，把数字或符号分配给现实世界实体的属性的过程，是任意一个工程过程中的重要元素。测度可以帮助人们更好地理解物质的属性，人们可以运用测度来评价工程化产品或系统的质量。软件工程和其他工程学科不一样，不是建立在物理基本定量定律上，不采用绝对测度，而是用一套间接测度方法来表示软件的质量。

（3）度量

建立在多次测度的基础上，对系统、部件或过程的某一特性所具有的程度进行的量化测量，如软件质量度量、可靠性度量等。

（4）指标

指标是一个度量或多个度量的组合，对同一类事物的多次度量的一个横向比较。它可以对软件产品、过程或资源提供更深入的理解。例如，有 4 个小组共同完成一个软件项目，每个小组都采用自行选择的评审类型进行技术评审。管理者检查“每小时每人所发现的错误数”这一度量结果时发现，采用正式技术评审方法的两个小组比另外两个小组高出 40%。假定 4 个小组的其他参数都一样，这就给管理者提供了一个指标：正式技术评审方法比其他技术评审方法效率更高。这样，管理者可以建议所有小组都采用正式技术评审方法。由此可见，度量和指标能提供更深入的理解，使项目管理者和开发者都在及时调整开发过程、

① Roger S. Pressman. 软件工程实践者的研究方法. 梅宏译. 北京：机械工业出版社，2002

项目或产品等方面做出更正确的决策。

（5）估算

对软件产品、过程资源等使用历史资料或经验公式等进行预测，如对工作量、成本、完成期限等。估算多用于立项、签订合同、制订计划等。

10.1.2 软件度量的分类

软件度量可以分为直接度量和间接度量两类。

（1）直接度量

即对不依赖于其他属性的简单属性的度量，如软件的模块数、程序的代码行数、操作符的个数、工作量、成本等。

（2）间接度量

对涉及若干其他属性的软件要素、准则或属性的度量，必须通过建立一定的度量方法或模型才能间接推断得到，如软件的功能性、复杂性、可靠性、可维护性等。

软件度量系统还可以划分为两个侧面：一个包含面向规模的度量、面向功能的度量和面向过程的度量；另一个包含生产率度量、质量度量和技术度量，如图10-1所示。

这两种分类方式有交叉，如软件的模块数、程序的代码行数等又是属于面向规模的度量。

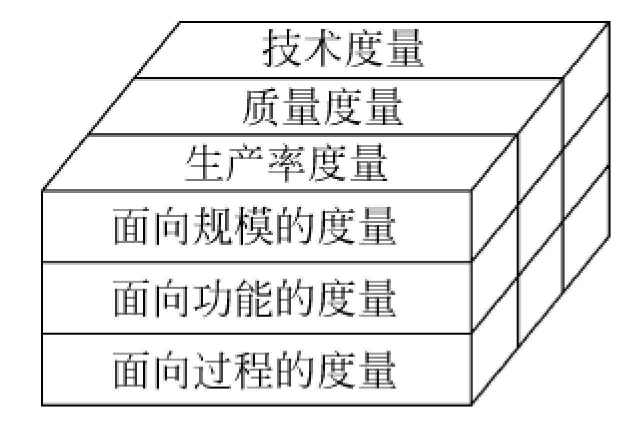


图10-1 软件度量的两个侧面

10.1.3 软件度量的目的

运用软件度量，能更好地理解所建立的模型的属性，更为重要的是，度量的结果可以帮助人们评价所建立的工程化产品或系统的质量。

总的来说，度量的目标是：

- 更好地理解产品质量。
- 评估过程的效率。
- 改善在项目级别完成的工作的质量。

这些目标都很重要，但是对软件工程师而言，最重要的是产品质量。

10.1.4 有效软件度量的属性

对于计算机软件，已经提出过很多的度量方法，但不是所有的都有实际意义，有的太复杂了，有的太深奥了，有的则违反了高质量软件的实际的基本概念。

Ejiogu定义了一组有效软件度量应该包含的属性，由其导出来的度量及测度如下[①]。

- 简单的和可计算的。如何导出度量值才是相对简单的，并且它的计算不应该要求过多的工作量和时间。
- 经验和直觉上有说服力的。度量应该满足工程师对于所考虑的产品的直觉概念（如一个测度模块内聚性的度量值应该随着内聚度的提高而提高）。

---

① Ejiogu L. Software Engineering with Formal Metrics. QED Publishing，1991

- 一致和客观的。度量应该总是产生无二义的结果，即用不同的方法对同一个属性进行度量时应该是一致的。
- 在单位和量纲的使用上是一致的。度量的数学计算应该使用不会导致奇异单位组合的测度。如把项目组的人数乘以编程语言变量就是一个直觉上没有说服力的单位的组合。
- 编程语言独立。度量应该基于分析模型、设计模型或程序本身的结构，不应该依赖于不同的编程语言的句法和语法，但和语言的风格相关，如对结构化的度量就不能采用面向对象的度量方法。
- 高质量反馈的有效机制。度量应该给软件工程师提供能生产更高质量的最终产品的信息。

下面针对软件开发过程中的各个阶段进行软件度量的介绍。

10.2 分析模型的度量

10.2.1 基于功能点的度量

在第 4 章中已经提到功能点度量可以用来作为预测从分析模型得到的系统大小的手段。考虑 ATM 系统的分析模型，图 10-2 描述了 ATM 系统的取款功能的数据流图。

为了确定用于计算功能点度量所需的关键测度，对数据流图加以评价：

(1) 外部输入有 8 个：账号、选择信息、确认信息、金额(分为取款金额和转账金额)、日期、账号(转入账号)、密码。

(2) 外部输出有 3 个：与屏幕和语音接口的数据接口信息。

(3) 用户查询有 5 个：余额信息、密码、转出账户的信息、明细信息、出钞器余额。

(4) 有 2 个文件：即客户存储信息单和系统配置信息。

(5) 外部接口有 8 个：与出钞器、后台服务系统、凭条打印机和读卡器的所有数据接口信息。

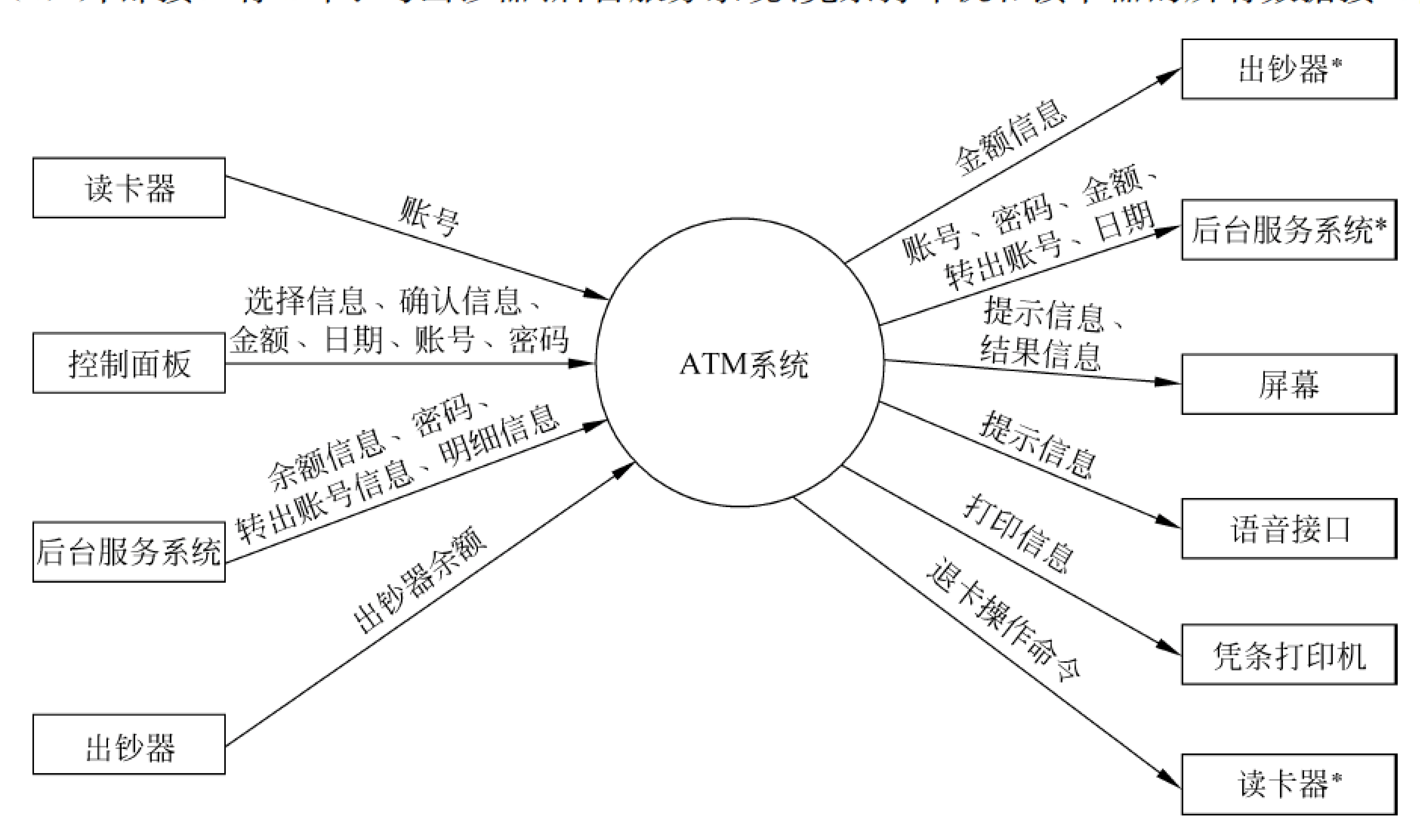


图 10-2 ATM 系统第 0 层数据流图

假定复杂度是平均的，这些数据及复杂度在图 10-3 中显示。

测量参数	计数		加权因子 简单	平均	复杂		
外部输入数	8	×	3	4	6	=	32
外部输出数	3	×	4	5	7	=	15
用户查询数	5	×	3	4	6	=	20
文件数	2	×	7	10	15	=	20
外部接口数	8	×	5	7	10	=	56
总计数值						→	143

图 10-3　功能点计算

图 10-3 中显示的总计数值还要用公式 FP＝总计数值×$(0.65+0.01\times\sum F_i)$调整，对于 ATM 系统，假定$\sum F_i$ 是 46（一个中度复杂的产品），则 $FP=143\times(0.65+0.01\times46)=158.7$。正如第 4 章提到的，得到功能点之后，依据历史数据，就能估算出软件项目的生产率、平均成本、代码出错率等参考值，可以给项目经理提供基于分析模型的初步估计的重要计划信息。

每个项目还可以得到如下一组基于功能点计算的度量。

设 FP 表示软件的功能点数；E 表示开发软件所需的工作量，单位为人月（PM）或人年（PY）；S 表示软件成本，单位为美元或元；N_d 表示缺陷数；Ne 表示错误个数；Pd 表示软件文档页数，则有：

- 软件生产率 P_f（即平均每人月开发的功能点数，以功能点/PM 为单位）

$$P_f = FP/E \tag{10-1}$$

- 出错率 EQR_f（即每功能点的平均错误数，以个/功能点为单位）

$$EQR_f = Ne/FP \tag{10-2}$$

- 缺陷率 DQR_f（即每功能点的平均缺陷数，以个/功能点为单位）

$$DQR_f = N_d/FP \tag{10-3}$$

- 平均成本 C_f（以美元/功能点或元/功能点为单位）

$$C_f = S/FP \tag{10-4}$$

- 软件的文档率 D_f（即平均每功能点的文档页数，以页/功能点为单位）

$$D_f = Pd/FP \tag{10-5}$$

10.2.2　bang 度量

和功能度量一样，bang 度量可以由分析模型得到对将要实现的软件大小的指示。为了计算 bang 度量，软件工程师必须首先评价一组原语，即在分析层次上不能再划分的分析模型元素[①]。原语的评价是通过评价分析模型和计数以下数据来决定的：

- 功能原语（FuP）。在数据流图中最低层次的变换（泡泡）的数量。

① Roger S. Pressman. 软件工程实践者的研究方法. 梅宏译. 北京：机械工业出版社，2002

- 数据元素(DE)。数据对象的不可再分割的属性的数量,数据元素必须在数据字典里出现。
- 对象(OB)。具有若干不同特性或属性的事物的表示,只封装数据,没有包括作用于数据的操作。
- 关系(RE)。数据对象之间的联系的数量。
- 状态(ST)。在状态转换图中用户可观察到的状态的数量。
- 转换(TR)。在状态转换图中状态转换的数量。

除了上述 6 个原语,还需要确定如下计数。

- 修改的手工功能原语(FuPm)。在系统边界之外且为了适应新系统而必须修改的功能。
- 输入数据元素(DEI)。
- 输出数据元素(DEO)。
- 保持的数据元素(DER)。被系统存储的数据元素。
- 数据记号(TC_i)。存在在第 i 个功能原语的边界上的数据记号(在一个功能原语内不可再分割的数据项)。
- 关系连接(RE_i)。在数据模型中连接第 i 个对象和其他对象的关系。

bang 度量是由 DeMacro 提出来的,包括两个度量:用于"功能强壮"系统的度量和用于"数据强壮"系统的度量。功能 bang 度量建立在数据流图中功能原语的数量的基础上,功能原语计数根据功能原语的类型和所使用的数据记号的数量进行加权;数据 bang 度量建立在实体关系模型中的实体数量的基础上,实体计数根据每个实体涉及的关系的数量进行加权。

RE/FuP<0.7 意味着是一个"功能强壮"的系统。

RE/FuP>1.5 意味着是一个"数据强壮"的系统。

0.8< RE/FuP<1.4 则意味着是混合型系统。

1) 计算功能 bang 度量可以用以下算法。

将 bang 初始化为 0;

当功能原语仍然可以估算时做如下事情:

(1) 计算对象 i 的关系的数量。

(2) 计算修正的功能原语的增量。

(3) 为功能原语声明类。

(4) 评价类,并且为类标出权值。

(5) bang=bang+权值×修正的功能原语增量。

通过确定在原语中有多少分离的记号是"可见的"来计算数据记号计数。修正的功能原语增量可以从 DeMarco 公开的表中得出,其中一个节选版本如表 10-1 所示。

表 10-1 修正的功能原语增量

数据记号	修正的功能原语增量	数据记号	修正的功能原语增量
2	1.0	15	29.3
5	5.8	20	43.2
10	16.6		

2）数据 bang 度量可以用以下算法进行计算。

将 bang 初始化为 0；

当数据模型中的对象仍然可以估算时做如下事情：

（1）计算在第 i 个功能原语的边界上的数据记号；

（2）计算修正的对象的增量；

（3）bang＝bang＋修正的对象的增量。

修正的对象增量可以从 DeMarco 公开的表中确定，其中一个节选版本如表 10-2 所示。

表 10-2　修正的对象增量

关系连接	修正的对象增量
1	1.0
3	4.0
6	9.0

一旦 bang 度量被计算后，可以借助过去的历史数据把它与大小和工作量等关联起来。DeMarco 建议一个组织建立它自己的两个表版本，用于从已完成的软件项目校准信息。

10.2.3　规格说明质量的度量

Davis 及其同事提出了一系列可以用来评价分析模型和相应的需求规格说明质量的特征：无二义性、完全性、正确性、可理解性、可验证性、内外部一致性、可完成性、简洁性、可跟踪性、可修改性、精确性和可复用性。Davis 等人建议每个特性用一到多个度量来表示，例如，在一个规格说明中有 n_r 个需求，则 $n_r = n_f + n_{nf}$，其中，n_f 是功能性需求的数目，n_{nf} 是非功能性（如性能）需求的数目。

为了评价无二义性，Davis 等人提出了一种基于评审者对每个需求解释的一致性的度量：$Q_1 = n_{ui}/n_r$，其中，n_{ui} 是所有评审者都做出相同解释的需求的数目。需求的二义性越低，Q 的值越接近 1。

功能性需求的完全性可以通过计算以下比例获得：

$$Q_2 = n_u/(n_i \times n_s)$$

其中，n_u 是独特的功能性需求的数目，n_i 是由规格说明定义或隐含的输入的个数，n_s 是被刻画的状态的个数。Q_2 比率测度了一个系统必需功能的百分比，但它没有考虑非功能性需求。我们把非功能性需求结合进来以求完全性，为此，需要考虑已经被确认的程度：

$$Q_3 = n_c/(n_c + n_{nv})$$

其中，n_c 是已经确认为正确的需求的数目，n_{nv} 是尚未被确认的需求的数目。

10.3　设计模型的度量

10.3.1　体系结构设计度量

体系结构设计度量集中于体系结构的特征上，强调体系结构的结构和模块的有效性，在某种意义上来说是黑盒的。

Card 和 Glass 定义了 3 个软件设计复杂度测度：结构复杂度、数据复杂度和系统复杂度[①]。

① Card D. N.，Glass R. L. Measuring Software Design Quality. Prentice-Hall，1990

模块 i 的结构复杂度 $S(i)$定义如下：

$$S(i) = f_{\text{out}}^2(i)$$

其中 $f_{\text{out}}(i)$是模块 i 的扇出数。

数据复杂度 $D(i)$定义如下：

$$D(i) = v(i)/[f_{\text{out}}(i) + 1]$$

其中 $v(i)$是模块 i 的输入输出变量的数目之和。

系统复杂度 $C(i)$定义为结构复杂度和数据复杂度之和：

$$C(i) = S(i) + D(i)$$

当结构复杂度或数据复杂度的值升高时，整个系统的复杂度也随之上升，导致集成和测试开销也跟着上升的可能性增大。

Henry 和 Kafura 提出的体系结构设计度量也使用了扇入扇出数：

$$HKM = length(i) \times [f_{\text{in}}(i) + f_{\text{out}}(i)]^2$$

其中，$length(i)$代表模块 i 的代码中语句的数目，$f_{\text{in}}(i)$是模块 i 的扇入数。Henry 和 Kafura 扩展了扇入扇出的概念，不仅包括了模块调用，还包括了模块 i 读取（扇入）或更新（扇出）另外模块的情况。如同 Card 和 Glass 提出的度量方法一样，Henry-Kafura 度量值的升高也会导致模块集成和测试工作量随之上升的可能性增大。

Fenton 建议使用简单的形态度量，如外形度量，这样不同的程序体系结构就可以用同一套简单的量纲加以比较。

例如 ATM 系统的体系结构，如图 10-4 所示，可以定义如下的度量：

$$size = n + a$$

其中，n 是结点，即模块的数目，a 是弧，即控制线的数目。对于 ATM 系统的体系结构

$$size = 14 + 13 = 27$$

深度＝从根结点到叶子结点的最长路径＝5。

宽度＝体系结构中的最宽的那一层的结点数＝5。

弧和结点的比率 $r=a/n$，测度了体系结构的连接密度，为体系结构的耦合提供了一个简单的指示。对于 ATM 系统的体系结构：

$$r = 13/14 = 0.93$$

美国空军系统司令部基于计算机程序的可测度设计特性开发了一组软件质量指示，使用了从数据和体系结构设计中获得的信息而导出了范围为 0～1 的设计结构质量指标 DSQI。为了计算 DSQI，必须获得以下值：

S_1 为在程序体系结构中定义的模块总数。

S_2 为正确功能依赖于数据输入源或产生要在其他地方使用的数据的模块数，通常控制模块不会被计入。

S_3 为正确功能依赖于前导处理的模块数。

S_4 为数据库中的条目数，包括数据对象和所有定义对象的属性。

S_5 为独特的数据库条目的总数。

S_6 为数据库段（不同的记录或个体对象）的数目。

S_7 为有单个入口和出口的模块数目（异常处理不被看做是多重出口）。

确定了 S_1～S_7 的值，就可以算出以下中间值：

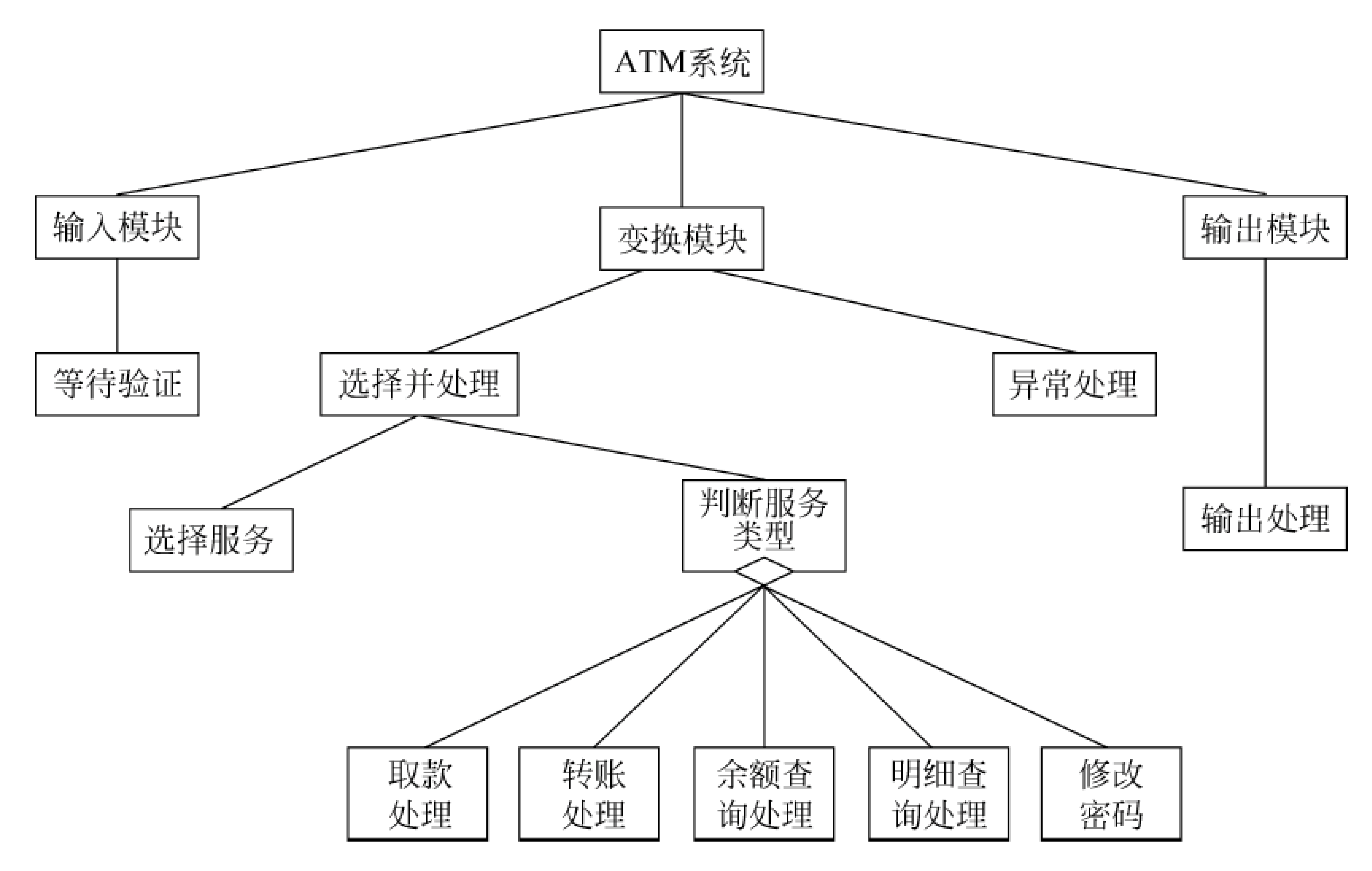


图 10-4　ATM 系统的软件体系结构

- 程序结构 D_1：如果体系结构设计是用一个独特的方法(如面向数据流或者面向对象设计)来开发，那么 $D_1=1$，否则 $D_1=0$。
- 模块独立性 D_2：$D_2=1-(S_2/S_1)$。
- 模块不依赖于前导处理 D_3：$D_3=1-(S_3/S_1)$。
- 数据库大小 D_4：$D_4=1-(S_5/S_4)$。
- 数据库划分 D_5：$D_5=1-(S_6/S_4)$。
- 模块出口/入口特性 D_6：$D_6=1-(S_7/S_1)$。

中间值确定后，可以用下面的方式来计算 DSQI：$\mathrm{DSQI}=\sum w_i D_i$，其中，$i$ 从 1 到 6，w_i 是每个中间值的重要性的相对权值，$\sum w_i=1$。如果所有 D_i 平均权值，则 $w_i=0.167$。

历史数据中的 DSQI 值和目前正在开发的设计相比较，如果 DSQI 明显低于平均值，意味着目前的设计需要进一步改进。当然，如果对现存的设计进行重要改动，这些改动对 DSQI 的影响也可以被计算出来。

10.3.2　构件级设计度量

构件级设计度量集中于软件构件的内部特性上，在某种意义上是白盒的，需要考虑模块的内部运作。一旦过程设计已经开发完，构件级设计度量就可以被应用，有时也可以延迟到源代码时再用。构件级设计度量包括模块内聚、耦合和复杂度 3 个测度。

1. 内聚度量

Bieman 和 Ott 定义了一组给模块内聚性提供指示的度量，以下 5 个概念和测度来定义①。

(1) 数据切片：对模块进行回溯走查所找到的影响走查开始处的模块位置的数据值。

① Bieman J. M., and Ott L. M. Measuring Functional Cohesion[J]. IEEE Trans. Software Engineering, 1994, 20(8): 308～320

（2）数据记号：为模块定义的变量。

（3）胶合记号：位于一个或多个数据切片的数据记号的集合。

（4）超胶合记号：在一个模块里对每个数据切片都公用的数据记号。

（5）黏度：一个胶合记号的相对黏度和它所绑定的数据切片的数目直接成比例。

Bieman 和 Ott 提出了强功能内聚 SFC、弱功能内聚 WFC 和附着性（胶合记号绑定数据切片的相对程度）的度量，这 3 个度量的取值范围为 0～1。当一个过程的输出多于一个并且没有展示任何由某一特定度量指示的内聚属性时值为 0。没有超胶合记号，即没有一个为所有数据切片所公用的数据记号的过程具有零强功能内聚，也就是说没有对所有输出有贡献的数据记号；一个没有胶合记号的过程，即没有对多于一个数据切片公用的数据记号（在具有多于一个的数据切片的过程中），展示了零弱功能内聚和零附着性，也就是没有对多个输出有贡献的数据记号。当 Bieman 和 Ott 度量取值为 1 时，出现了强功能内聚和附着性。强功能内聚度量 $\mathrm{SFC}(i)=SG(SA(i))/tokens(i)$，其中 $SG(SA(i))$ 指超胶合记号——位于模块 i 的所有数据切片数据记号的集合。当超胶合记号与模块 i 中所有记号的总和的比值上升时，模块的功能内聚性也增加了。

2. 耦合度量

模块耦合提供了对一个模块与其他模块、全局数据和外部环境的连接的指示。

Dhama 提出了一个耦合度量，包含数据和控制流耦合、全局耦合和环境耦合，计算模块耦合度量要对上述 3 种耦合类型的每一种进行定义①。

（1）数据和控制流耦合

d_i＝输入数据参数的个数

c_i＝输入控制参数的个数

d_o＝输出数据参数的个数

c_o＝输出控制参数的个数

（2）全局耦合

g_d＝用做数据的全局变量的个数

g_c＝用做控制的全局变量的个数

（3）环境耦合

w＝被调用模块的个数（扇出）

r＝调用所考虑的模块的个数（扇入）。

模块耦合度量

$$m_c = k/M$$

其中，$k=1$，$M=d_i+a\times c_i+d_o+b\times c_o+g_d+c\times g_c+w+r$，$a=b=c=2$，在不同情况下，$k$ 和 a、b、c 的值可以进行调整。m_c 的值越高，整体模块耦合度越低。

考虑耦合度最低的情况，一个模块只有一个单一的输入和输出参数，没有访问全局数据，并且只被一个模块所调用，那么 $m_c=1/(1+2\times0+1+2\times0+0+2\times0+1+0)=1/3=$

① Dhama H. Quantitative Models of Cohesion and Coupling in software[J]. Journal of Systems and software，1995，29(4)

0.33,意味着低耦合。再如,一个模块有 5 个输入和 5 个输出数据参数,相等数目的控制参数,访问了 10 个全局数据,且有 3 个扇入和 4 个扇出,则 $m_c=1/(5+2\times5+5+2\times5+10+2\times0+3+4)=1/47=0.02$,意味着高耦合。

为了让耦合度量随耦合度的升高而上升,我们将刚才的耦合度量改进为 $C=1-m_c$,其中 m_c 的取值范围为(0,0.33],改进后的耦合度量在最小值 0.66 到接近 1 的最大值之间非线性上升。

3. 复杂度度量

复杂度度量可以用来预计关于从源代码的自动分析得到的软件系统的可靠性和可维护性,也在软件项目中提供反馈以帮助控制设计活动,为测试和维护提供模块的详细信息以帮助指出潜在的不稳定的区域。

McCabe 认为程序控制流的复杂度度量很多都是基于流图的。流图可以从算法设计导出,是由结点和边构成的。图 10-5 给出了一个模块的流图设计,其中(a)图是用流程图表示的该模块的算法设计,由于过程设计中遇到复合条件,需要将复合条件中的每个条件分离出来,用独立的结点表示;(b)图就是将复合条件分离后的流程图;(c)图即该模块对应的流图。

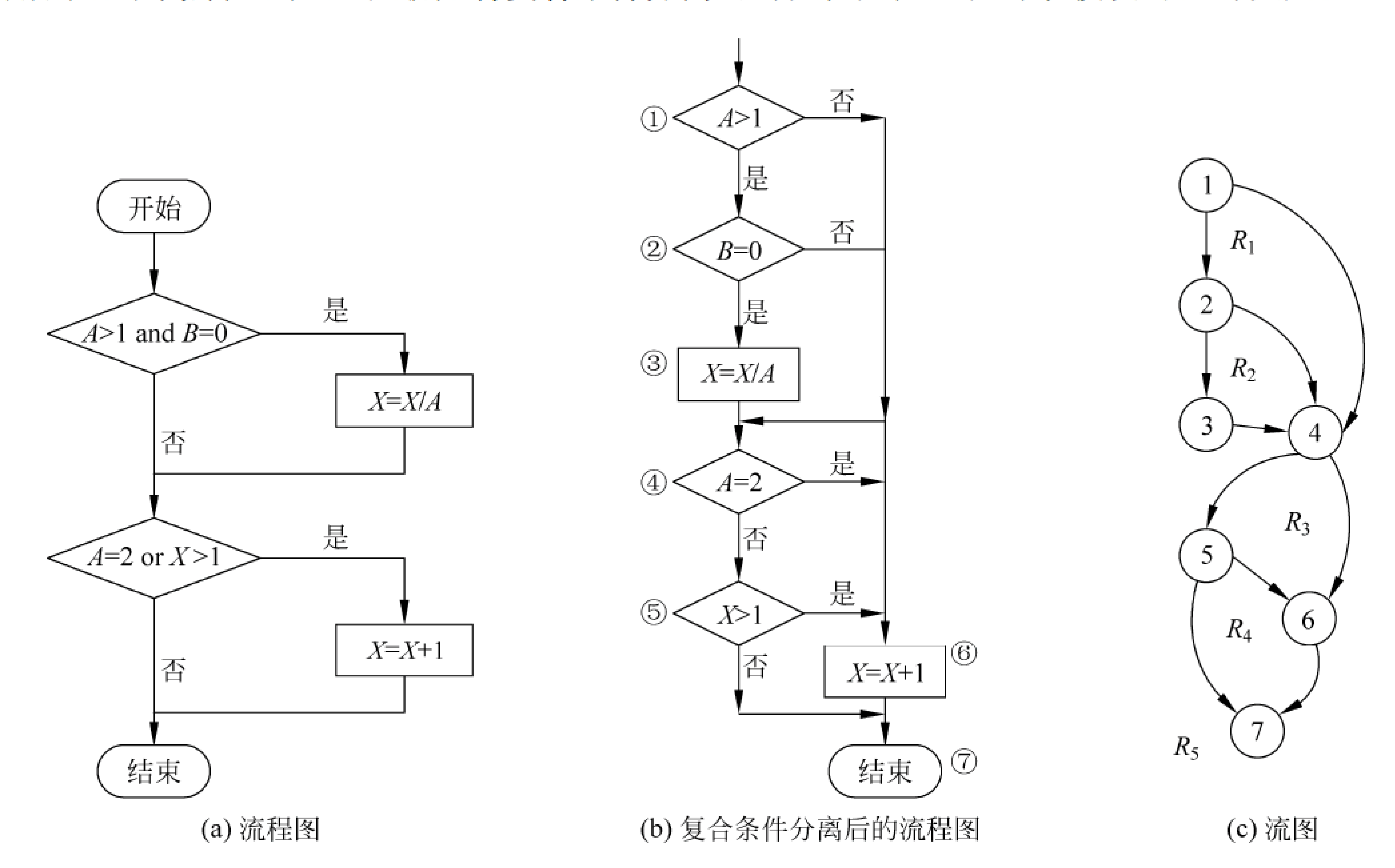


图 10-5 流图设计

复杂度度量的一种方法是计算流图的环复杂度,环复杂度定义了程序基本集的独立路径数量的上界。独立路径是指程序中至少引入一个新的处理语句集合或一个新条件的任一路径,即必须至少有一条语句在定义该路径之前不曾用到过。实际上基本集并不唯一,给定的过程设计可以派生出不同的基本集。可以用如下 3 种方法之一来计算环复杂度 $V(G)$。

(1) $V(G)$=流图中区域的数量,区域是指结点和边限定的区域,分为闭合区域和外部区域。图 10-5 中的流图有 4 个闭合区域,分别是结点 1、2、4 及它们之间的边围成的区域

R_1，结点 2、3、4 及它们之间的边围成的区域 R_2，结点 4、5、6 及它们之间的边围成的区域 R_3，结点 5、6、7 及它们之间的边围成的区域 R_4；还有 1 个外部区域 R_5，所以

$$V(G) = 4 + 1 = 5$$

(2) $V(G)=E-N+2$，其中 E 是流图中边的数目，N 是流图中结点的数目。图 10-5 的流图有 10 条边、7 个结点，所以

$$V(G) = 10 - 7 + 2 = 5$$

(3) $V(G)=P+1$，其中 P 是流图中的判定结点的数目，判定结点是指包含了条件的结点。图 10-5 的流图中结点 1、2、4、5 是判定结点，所以

$$V(G) = 4 + 1 = 5$$

程序结构的复杂性度量值 $V(G)$ 取决于程序控制流的复杂程度。当程序内的分支数和循环数增加时，$V(G)$ 值将随之增加，即程序的复杂性增大。$V(G)$ 还可以作为程序规模的定量指标，$V(G)$ 值越高的程序往往是越复杂、越容易出问题的程序。因此，McCabe 建议模块规模以 $V(G)\leqslant 10$ 为宜。

Halstead 给出了称为文本复杂性度量的模型，它是根据源代码中的操作符和操作数的个数来度量程序的复杂程度。

10.4 源代码度量

在第 4 章中提到了面向规模的估算，其中一种方法的衡量尺度就是代码行，通过对代码行的计数得到 LOC。LOC 的获取很简单，但是 LOC 对系统的度量依赖于具体的语言和算法。为此，可以通过计算代码所包含的信息量进行度量，避免不同语言对度量带来的影响。

程序可以看成是由操作符和操作数组成的符号序列，操作符是指程序中出现的语法符号，如＋、－、if-then-else、while 等；操作数是操作对象，如程序中定义或使用的变量、常量、数组、指针等。Halstead 的软件科学理论把定量定律赋予了计算机软件开发，软件科学使用的一组基本测度在代码产生后或设计完成后对代码估算得到：

n_1＝在一个程序中出现的不同操作符的个数

n_2＝在一个程序中出现的不同操作数的个数

N_1＝操作符出现的总数

N_2＝操作数出现的总数

Halstead 使用基本测度来计算以下几个度量：

程序长度 $N=n_1\log_2 n_1+n_2\log_2 n_2$

程序体积 $V=N\log_2(n_1+n_2)$，V 代表了写一个程序所需的信息量，它会随着编程语言的不同而不同。

下面给出了一段源代码，对它进行 Halstead 度量。

```
addend = 0;
j = 0;
while (j< = 10)
{
```

```
    addend = addend * 10 + 3;
}
print(addend);
```

其中：

操作符：=,; ,while,<=, * ,+,print,( )

操作数：addend,0,j,10,3

因此,$n_1=8$,$n_2=5$

长度 $N=n_1\log_2 n_1+n_2\log_2 n_2=8\log_2 8+5\log_2 5=8\times 3+5\times 2.32=35.6$。

理论上,算法存在着最小体积,Halstead 将体积比率 L 定义为程序最简洁形式的体积与实际程序体积的比,即 $L=2/n_1\times n_2/N_2$,显然 L 总是小于 1。

除了已经介绍过的度量以外,每个项目还可以得到如下一组简单的源代码的度量。

设 L 表示软件的代码行数,单位为 KLOC; E 表示开发软件所需的工作量,单位为人月(PM)或人年(PY); S 表示软件成本,单位为美元或元; N_d 表示缺陷数; Ne 表示错误个数; Pd 表示软件文档页数,则有：

- 软件生产率 P_l(即平均每人月开发的代码行数,以 LOC/PM 为单位)

$$P_l = L/E \tag{10-6}$$

- 每千行代码的错误数 EQR_l(即每千行代码的平均错误数,以个/KLOC 为单位)

$$EQR_l = Ne/L \tag{10-7}$$

- 每千行代码的缺陷数 DQR_l(即每千行代码的平均缺陷数,以个/KLOC 为单位)

$$DQR_l = N_d/L \tag{10-8}$$

- 每千行代码的成本 C_l(以美元/LOC 或元/LOC 为单位)

$$C_l = S/L \tag{10-9}$$

- 每千行代码的文档页数 D_l(即平均每千行代码的文档页数,以页/KLOC 为单位)

$$D_l = Pd/L \tag{10-10}$$

此外,还能计算出其他有意义的度量：

- 每人月错误数$=Ne/E$
- 每页文档的成本

【例 10-1】 已知有一个国外典型的软件项目的记录,其代码行数=20.2KLOC,工作量 E=43PM,成本 S=314 000 美元,错误数 Ne=64,缺陷数 N_d=6,文档页数 Pd=1050 页。试计算开发该软件项目的生产率 P_l、平均成本 C_l、代码出错率 EQR_l、缺陷率 DQR_l 和文档率 D_l。

解：根据给出的已知数据,可得：

$$P_l = L/E = 20.2\text{KLOC}/43\text{PM} = 0.47\text{KLOC/PM} = 470\text{LOC/PM}$$

$$C_l = S/L = 314\,000\text{ 美元}/20.2\text{KLOC} = 15.54\text{ 美元}/\text{LOC}$$

$$EQR_l = Ne/L = 64\text{ 个}/20.2\text{KLOC} = 3.17\text{ 个}/\text{KLOC}$$

$$DQR_l = N_d/L = 6\text{ 个}/20.2\text{KLOC} = 0.3\text{ 个}/\text{KLOC}$$

$$D_l = Pd/L = 1050\text{ 页}/20.2\text{KLOC} = 51.98\text{ 页}/\text{KLOC}$$

10.5 对测试的度量

软件测试的度量大部分都集中在测试的过程,而不是测试本身的技术特性。通常测试案例的设计和执行都依赖于分析、设计和代码度量的指导。

基于功能点的度量可以用来作为整体测试工作量的指示器,历史数据中的不同项目层次特性,如测试工作量和时间、未发现的错误、产生的测试案例的数目等都可以收集起来和项目的 FP 数相关联,还可以为将来的项目的这些特征预测估算值。

bang 度量中的功能原语、数据元素、对象、关系、状态和变迁可用来预测软件黑盒、白盒测试的数目和类型。

体系结构设计度量提供了和集成测试相关的难易信息以及对专用测试软件的需求。如上面提到的环复杂度在基本路径测试中要用到,还可以用来判断哪些模块作为广泛的单元测试的候选,环复杂度高的模块可能比较容易出错,测试者应该在这种模块集成进系统前花费比较多的工作量来发现模块中的错误。测试工作量 e 也可以用 Halstead 度量来估算:$e=V/PL$,其中,V 是程序体积,程序层次 $PL=1/[(n_1/2)(N_2/n_2)]$。将被分配给模块 k 的测试工作量百分比$=e(k)/\sum e(i)$,其中,工作量 $e(k)$可根据刚才的公式计算出来,$\sum e(i)$是系统所有模块的工作量总和。

测试时,测试宽度、深度和错误轮廓的测度构成了对测试完全性的指示。测试宽度的测度指示了多少需求已经被测试过;测试深度是被测试覆盖的独立路径占程序中的基本路径总数的百分比,其中基本路径的数目可以通过累加所有程序模块的环复杂度得到;当收集测试中的错误数据时,错误轮廓可以用来对未发现的错误进行优先级和分类处理,优先级指明问题的严重性,错误类别可以对错误进行统计分析。

10.6 对维护的度量

IEEE Std. 982.1—1988 提出软件成熟度指标 SMI,它提供了对软件产品稳定性的指示(基于为每个产品增量发布而做的变更),可以用来度量维护活动。根据以下信息来计算软件成熟度[①]:

M_T=当前发布中的模块数

F_c=当前发布中已经变更的模块数

F_a=当前发布中已经增加的模块数

F_d=当前发布中已删除的前一发布中的模块数

$$\text{SMI}=[M_T-(F_a+F_c+F_d)]/M_T$$

当 SMI 接近 1.0 时,产品开始稳定。SMI 也可以用做计划软件维护活动的度量。产生一个软件产品的发布的平均时间和 SMI 关联,可以开发一个维护工作量的经验模型。

---

① Roger S. Pressman. 软件工程实践者的研究方法. 梅宏译. 北京:机械工业出版社,2002

10.7 软件质量的度量

在第 5 章中已经学习了 McCall 软件质量模型，介绍了 11 个软件质量要素。McCall 等人通过确定影响软件质量要素的属性定义了 21 个软件质量要素的评价准则，这些准则可以间接测量软件质量要素，从而可以度量整个软件质量。下面介绍这 21 个准则[①]：

(1) 完全性：软件系统不丢失任何重要成分完全实现所需功能的程度。

(2) 一致性：在软件开发项目中一致的设计和文档技术的使用。包括整个软件系统的各个模块应使用一致的概念、符号和术语；程序内部接口应该保持一致；软件与环境的接口应该保持一致；系统规格说明与系统行为应该保持一致；用于形式化规格说明的公理系统应该保持一致。

(3) 模块化：程序部件的功能独立性，指的是将一个程序划分为若干个模块，每个模块完成一个子功能，把这些模块组装成一个整体，即可实现该程序指定的功能，而且每个模块是相对独立的成分，是独立的编程单位，以实现信息隐藏和抽象。

(4) 可追踪性：根据软件需求对设计、代码进行正向追踪，或根据代码、设计对软件需求进行逆向追踪的能力。

(5) 可审查性：检查软件需求、文档、过程、标准等是否一致的难易程度。

(6) 准确性：计算和控制的精确程度。

(7) 简明性：程序源代码的紧凑程度，以代码行数来评价。

(8) 通信通用性：使用标准接口、协议和带宽的程度。

(9) 数据通用性：在程序中使用标准数据结构和类型的程度。

(10) 容错性：在各种异常情况下软件能继续运行的能力，以及程序遇到错误时所造成的损失。

(11) 执行效率：软件运行的效率。

(12) 可扩充性：结构、数据、过程等设计可以扩展的程度。

(13) 通用性：程序潜在的应用领域的多少。

(14) 硬件独立性：软件与其运行的硬件环境无关的程度。

(15) 自检测性：程序监视自身运行并标识错误的程度。

(16) 可操作性：操作该软件的难易程度。

(17) 安全性：控制或保护程序和数据不被破坏、非法访问等机制的能力。

(18) 自文档化：源代码提供自身说明文档的程度。

(19) 简单性：程序易于理解的程度。

(20) 软件独立性：软件与非标准编程语言特征、操作系统特征等软件环境约束无关的程度。

(21) 易培训性：软件对使用它的新用户的支持程度。

11 个软件质量要素和 21 个评价准则之间的关系如表 10-3 所示。

---

① 曹哲，高诚，等. 软件工程. 北京：中国水利水电出版社，2004

表 10-3 质量要素与评价准则的关系

质量要素＼评价准则	可追踪性	完全性	一致性	容错性	准确性	简单性	可操作性	执行效率	可审查性	自检测性	安全性	数据通用性	可扩充性	通用性	硬件独立性	简明性	通信通用性	自文档性	软件独立性	易培训性	模块化
正确性	√	√	√																		
可靠性			√	√	√	√															
效 率						√	√	√													
完整性									√	√	√										
可使用性							√													√	
可维护性			√			√				√						√		√			√
可测试性									√	√						√		√			√
灵活性			√			√							√	√		√		√			√
可移植性														√	√			√	√		√
复用性														√	√			√	√		√
互连性												√			√		√				√

由表 10-3 给出的软件质量要素与评价准则之间的关系可以通过测量评价准则来间接度量软件质量要素。第 j 种软件质量要素 $F_j(j=1,2,\cdots,11)$ 的计算公式为

$$F_j = \sum_{k=1}^{21} C_{jk} M_k$$

其中：M_k 是第 j 种软件质量要素 F_j 对第 k 种评价准则的测量值，该值靠经验给出。McCall 将每个准则划分为 0～10 级，M_k 的值可以在 0，0.1，0.2，…，1.0 中取一个；C_{jk} 为加权系数，满足 $\sum_{k=1}^{21} C_{jk} = 1, C_{jk} \geqslant 0, C_{jk} = 0$ 表示质量要素与第 k 种评价准则无关。

例如，要度量某软件的质量要素 F_2（可靠性），假设 $C_{23}=0.1, C_{24}=0.3, C_{25}=0.4, C_{26}=0.2$，其余的 $C_{2k}=0$，$M_3=0.7, M_4=0.6, M_5=0.5, M_6=0.8$，则可靠性的度量值为：

$$F_2 = C_{23}M_3 + C_{24}M_4 + C_{25}M_5 + C_{26}M_6 = 0.1\times0.7 + 0.3\times0.6 + 0.4\times0.5 + 0.2\times0.8 = 0.61$$

除了 McCall 的软件质量度量模型外，还有很多软件工程组织和专家在软件质量度量方面做了大量工作，但正确性、可维护性、完整性、可靠性和可用性往往能为项目提供有用的指标，下面就这些方面来介绍软件质量度量的一些方法。

1. 软件正确性

一个程序必须能够正确地执行，否则对于用户而言就没有价值了。正确性包含两方面，一方面是指软件完成所要求的功能的程度，另一方面指的是软件没有错误。可以用本章 10.2.1 小节和 10.4 节中提到的代码缺陷率 DQR_f 和每千行代码的缺陷数 DQR_l 度量软件完成所要求功能的程度，用 10.2.1 小节和 10.4 节中提到的代码出错率 EQR_f 和每千行代码的错误数 EQR_l 来度量软件的错误。

2. 软件可维护性

在软件工程的整个生命周期中，维护需要非常多的工作量。软件的可维护性是指为满

足用户新的要求，或当环境发生了变化，或运行中发现了新的错误时，对软件进行相应诊断和修改的容易程度。目前还没有一种方法能够对软件的可维护性进行直接度量，只能采用间接测量。有一种简单的面向时间的度量，称为平均变更时间 MTTC，包括分析变更请求、设计合适的修改方案、实现变更并进行测试，以及将该变更发布给全部用户所花的时间。一般情况下，MTTC 值越低，可维护性越好。

3. 软件完整性

软件的完整性是指为了某一目的而保护数据，避免它受到偶然的，或有意地破坏、改动或遗失的能力。在网络时代，软件完整性变得越来越重要，它测量的是一个系统对安全性攻击(包括对程序、数据和文档的攻击)的抵抗能力。

为了测量完整性，引入另外两个属性：危险性和安全性。危险性指一个特定类型的攻击在给定时间内发生的概率，安全性是指一个特定类型的攻击被击退的概率。那么系统的完整性可以定义为：完整性$=\sum[1-$危险性$\times(1-$安全性$)]$。

例如，一个系统的危险性(发生攻击的可能性)是 0.2，安全性(击退攻击的可能性)是 0.9，则系统的完整性是 0.98，说明该系统的完整性很高。

4. 软件可靠性

软件可靠性是一个重要的软件质量要素，尤其是对于实时、嵌入式计算机系统。软件可靠性和其他软件质量要素不同，它可以直接度量，也可以用历史数据或开发数据计算。

在第 5 章中已经介绍了软件可靠性的定义，可以用在给定时间间隔内程序按规格说明成功运行的概率来衡量。用随机变量 t 表示发生故障的时刻，$t\in[0,\infty]$；函数 $f(t)$ 是随机变量 t 的概率密度函数，$F(t)$ 表示分布函数；$P(0\leqslant t\leqslant t_1)$ 表示从初始时刻到 t_1 时刻程序发生故障的概率。假设初始时刻程序运行正常，即 $F(0)=0$，则

$$F(t)=\int_0^t f(x)\mathrm{d}x \tag{10-11}$$

$$f(t)=\frac{\mathrm{d}F(t)}{\mathrm{d}t} \tag{10-12}$$

用 $P_f(t_1)$ 表示从 0 时刻到 t_1 时刻程序发生故障的概率，则有

$$P_f(t_1)=P(0\leqslant t\leqslant t_1)=F(t_1)-F(0)=F(t_1) \tag{10-13}$$

那么程序运行成功的概率

$$R(t)=1-P_f(t)=1-F(t)=1-\int_0^t f(x)\mathrm{d}x$$

可以看出，当错误数一定时，程序运行的时间越长，发生故障的次数越多，软件的可靠性越低。

软件可靠性可以用软件平均无故障时间来估算。软件平均无故障时间 MTTF 是指系统可以无故障持续运行的时间，它是可靠性度量的一个重要参数，往往作为一个重要的质量指标。下面介绍两种情况下的 MTTF 的估算方法。

(1) 当软件故障率 λ 为常数时。假设程序运行 H 小时，共发生 r 次故障，则软件故障率

$$\lambda\approx r/H$$

可得

$$\text{MTTF} = 1/\lambda = H/r$$

（2）软件故障率与程序剩余的错误数成正比。假设 I_T 为程序代码长度；E_T 为测试前程序中剩余的错误总数；$E_c(\tau)$为$[0,\tau]$区间内改正的错误数；$E_r(\tau)$为在 τ 时刻程序中剩余的错误数，其中 τ 为调试和排错时间，有

$$E_r(\tau) = E_T - E_c(\tau) \tag{10-14}$$

式子的两边同时除以 I_T，则

$$E_r(\tau)/I_T = E_T/I_T - E_c(\tau)/I_T \tag{10-15}$$

令 $\varepsilon_r(\tau)=E_r(\tau)/I_T$，$\varepsilon_T=E_T/I_T$，$\varepsilon_c(\tau)=E_c(\tau)/I_T$

于是有

$$\varepsilon_r(\tau) = \varepsilon_T - \varepsilon_c(\tau) \tag{10-16}$$

由于软件故障率 λ 与错误数成正比，所以

$$\lambda = k\varepsilon_r(\tau) = k[\varepsilon_T - \varepsilon_c(\tau)] \tag{10-17}$$

$$\text{MTTF} = 1/\lambda = \frac{1}{k[\varepsilon_T - \varepsilon_c(\tau)]} \tag{10-18}$$

比例因子 k 可以通过实验测试和统计的方法来估算。设进行 n 次软件排错实验，时间区间为$[0,\tau_j]$，到 τ_j 时刻为止，共排除了 $E_c(\tau_j)$个错误，而在时间区间$[0,\tau_j]$内，程序运行了 H_j 小时，出现了 r_j 个错误，$j=1,2,\cdots,n$。此时 k 的估计值为：

$$\hat{k} = \frac{\sum_{j=1}^{n} r_j}{H \cdot \sum_{j=1}^{n}[\varepsilon_T - \varepsilon_c(\tau_j)]} \tag{10-19}$$

当 $n=1$ 时，

$$k = \frac{r}{H[\varepsilon_T - \varepsilon_c(\tau_j)]}$$

当 $n=2$ 时，

$$k = \frac{r_1 + r_2}{H_1[\varepsilon_T - \varepsilon_c(\tau_1)] + H_2[\varepsilon_T - \varepsilon_c(\tau_2)]}$$

k 的值估算出来以后，就可以估算 MTTF 的值。随着软件测试和维护工作的不断进展，软件中剩余的错误不断减少，故障率 λ 不断降低，MTTF 不断增大，软件可靠性也随之不断提高。对于确定的 τ 值，$\lambda=k\varepsilon_r(\tau)$为常数，经过$[0,\tau]$区间的排错后，软件可靠性估算为：

$$R(t) = e^{-k\varepsilon_r(\tau)t} = e^{-\frac{t}{\text{MTTF}}} \tag{10-20}$$

其中：时间参数 τ 以月计，表示对程序调试和维护的时间；$t\in[0,\tau]$，以小时计，表示程序运行的时间。式(10-20)表示经过 τ 个月的调试后所达到的软件可靠性。

在实际应用中，可以利用概率统计的方法进行比较简单的计算：

$R(n)$表示在特定时间 n 内软件不失败的概率，n 的单位可以是天；$F(n)=1-R(n)$表示特定时间内软件失败的概率。如果已知错误率，即错误间隔时间的倒数，就可以算出 $F(1)$。那么可靠性 $R(n)=R^n(1)=[1-F(1)]^n$。

例如，平均每两天发生一个错误，求系统在 1、2、3 和 4 天的不失败概率。

平均每两天发生一个错误，则

$$F(1) = 0.5$$
$$R(1) = 1 - F(1) = 0.5$$
$$R(2) = R(1)^2 = 0.5^2 = 0.25$$
$$R(3) = R(1)^3 = 0.5^3 = 0.125$$
$$R(4) = R(1)^3 = 0.5^3 = 0.0625$$

又如，某软件在10天的测试中，测试100次出现5次错误，那么下一天的软件可靠性的估计值是

$$F(1) = 5/100 = 0.05$$

按照以往记录，假设每天测试10次(最近10天的平均值)，则

$$R(10) = R^{10}(1) = 0.95^{10} = 0.598$$

5. 软件可用性

软件可用性是程序根据需求，在给定的环境和时间内其可运行的概率①。

$$可用性 = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \times 100\%$$

其中，MTTF和MTTR分别为平均无故障时间和平均故障修复时间。实际上，公式的分母部分就是平均故障间隔时间MTBF=MTTF+MTTR，它是指相邻两次故障发生的时间间隔。

如果MTTR忽略不计，并且假设在0时刻到t时刻内，系统无故障，则系统是可靠的，这就是系统的可靠性，而时间t就是可用性。

可用性是软件可维护性的一种间接度量。

本章小结

测量能使管理者和开发者改进软件过程，辅助进行软件项目的计划、跟踪及控制，评估生成的软件的质量。而对过程、项目及产品的特定属性的测量可用来计算软件度量，分析这些度量可以获得指导管理及技术行为的指标，也为创建有效的分析模型、设计模型、可靠的代码和完全的测试提供必要的理解。

分析模型的度量集中于3个域——功能、数据、行为，功能点度量和bang度量给评价分析模型提供了定量的方法。其中，在功能点度量的基础上，还给出了生产率、出错率、平均成本等的度量。

设计度量分别考虑了体系结构的高层次和构件层次两方面。高层次度量考虑了体系结构的特征，如系统复杂度、扇入、扇出等，以及外形度量，如深度、宽度等；构件级设计度量通过内聚、耦合和复杂度进行度量。

Halstead提供了源代码级度量，通过代码中出现的操作符和操作数的数量，对代码所包含的信息量进行度量。另外，基于代码行规模的度量，给出了一组简单的源代码的度量，包括生产率、代码错误率、成本等。

① 杨文龙，古天龙. 软件工程. 北京：电子工业出版社，2007

软件质量包括了许多不同的产品和过程因素及其相关的度量，侧重于正确性、可维护性、完整性、可靠性和可用性等方面。

思考与练习

1. 请用自己的语言描述测量、测度、度量、指标的不同，并举出具体实例加以说明。

2. 在一个小系统中，有3个外部输入、2个外部查询、1个内部逻辑文件、2个外部输出和4个外部接口文件，所有这些数据属于平均复杂度，整个系统相对简单。试计算该系统的FP。

3. 下面给出了一段源代码，对它进行Halstead度量。

```
x = 4;
y = x;
if (x>=10) y = 3 * x - 11;
else if (x>=1) y = 2 * x - 1;
print(y);
```

4. 用本章10.3.2小节中介绍的3种方法计算如图10-6所示的程序流程图的环复杂度。

5. R(200)＝0.93表示假设有100个相同的系统同时启动运行，运行到200小时这一时刻，其中有93个处于正常运行状态，7个出现故障，等待修复。求R(201)。

6. 假定测试反映了操作情况，若在200个测试样例中有10个错误，试计算软件系统的可靠性。

7. 某系统体系结构图如图10-7所示，求该体系结构的深度和宽度，以及模块m的扇出、r的扇入数。

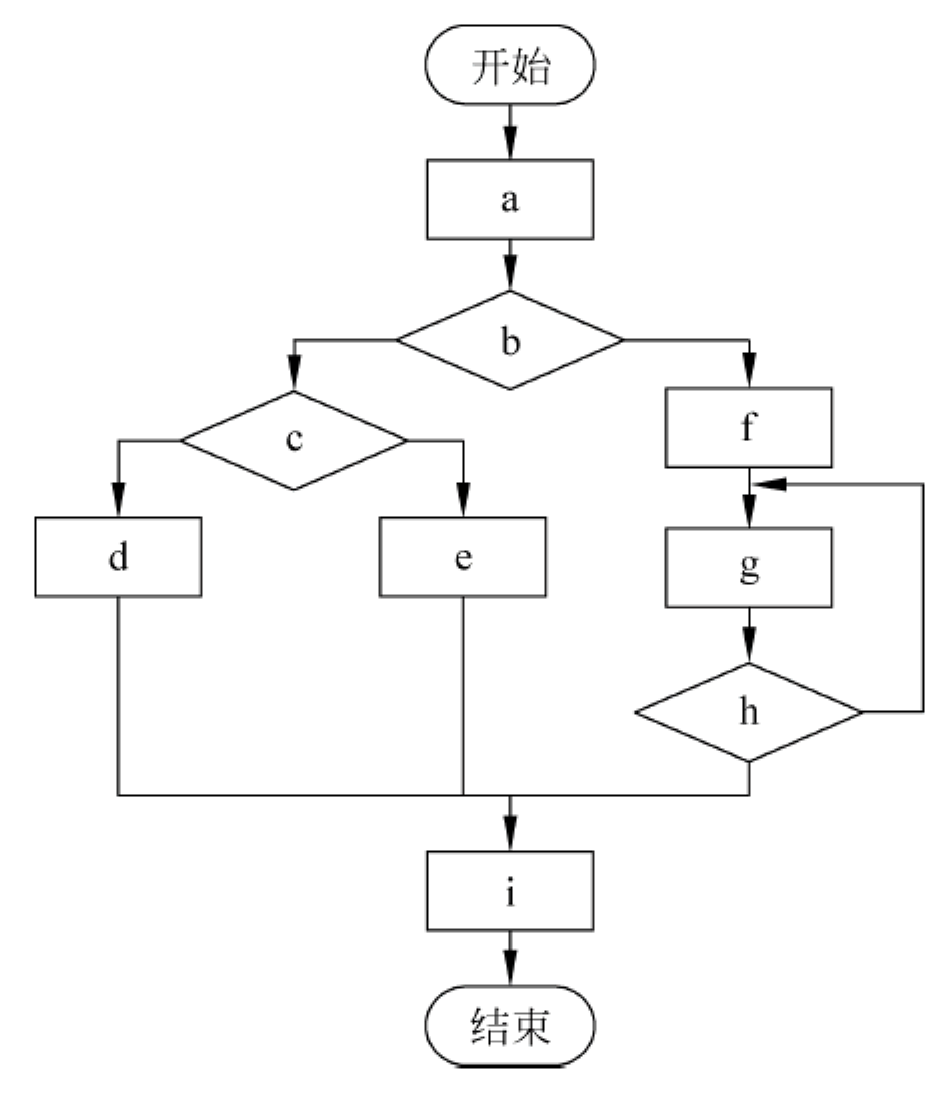


图10-6　流程图

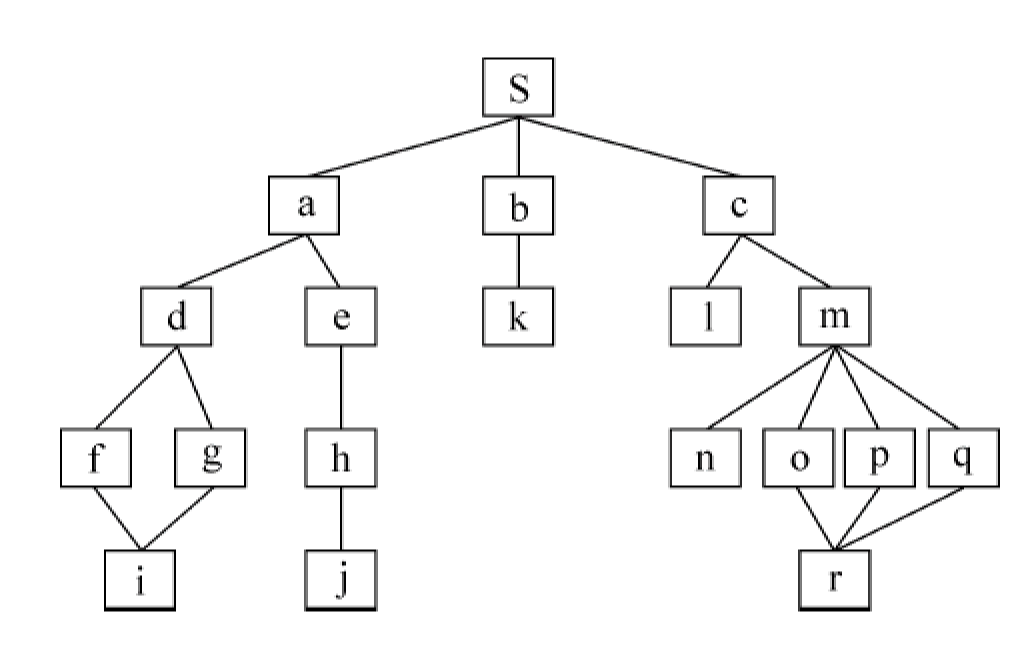


图10-7　某系统体系结构图

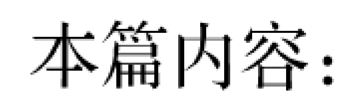
8. 试计算如表 10-4 所示的 3 个项目各自的生产率 P_l、平均成本 C_l、代码出错率 EQR_l 和文档率 D_l。

表 10-4　项目信息

项目	工作量/PM	成本/万美元	代码行 KLOC	文档页数 Pd	错误数 Ne
Alpha	24	16.8	12.1	365	29
Beta	62	44.0	27.2	1224	86
Gamma	43	31.4	20.2	1050	64

第3篇

面向对象的开发方法



本篇内容：

- 面向对象概述
- 面向对象分析
- 面向对象设计
- 面向对象测试
- 面向对象系统的技术度量

第11章 面向对象概述

今天大部分的软件都很像上百万块砖堆叠在一起组成的金字塔，缺乏结构完整性，只能靠强力和成千上万的奴隶完成。

——Alan Kay(图灵奖获得者，面向对象创始人之一)

对象是世界中的物体在人脑中的映像，是人类最朴素的认知世界的思维方式——任何事物都可以看成是对象。我们认识一种新的物体时，如看到某一绿色植物，称为“树”，于是在我们的意识当中就形成了树的概念。这个概念会一直存在于我们的思维当中，并不会因为这棵树被砍掉而消失。这个概念就是现实世界当中的物体在我们意识当中的映像。人们认识世界就是这样直观、朴素，所有的人类活动都是基于对对象进行创建、分类、组合和操作的。从古至今，人类在认识世界的过程中，也是以对象为个体逐个区分，理解对象在系统中的相互关系。因此，对于人类而言，生活在对象的世界中，这些对象存在于自然、生活周围、日常工作甚至于思维中。因此，软件开发提出面向对象的观点毫不为奇。这种抽象的建模思想，可以更好地帮助人们理解和分析客观世界①。

11.1 面向对象的基本思想

面向对象开发的思想与人类习惯的思维方法一致，它从对象出发去认识客观世界，分析问题域，如实地描述问题域中事物之间存在的各种关系。

传统的程序设计技术是面向过程的设计方法，这种方法以算法为核心，把数据和过程分为相互独立的部分，数据代表问题空间中的客体，程序代码则用于处理这些数据。这种把数据和代码作为分离的思想，是计算机处理数据思路的直接体现，因为在计算机内部，数据和程序是各自独立存放的。但这样做时总存在使用正确的程序模块处理错误的数据，或使用错误的程序模块处理正确的数据的危险。使数据与对应的程序模块保持一致是程序员的一个沉重负担。特别是在多人分工合作开发一个大型软件系统的过程中，如果负责设计数据结构的程序员中途改变了某个数据的结构而又没有及时通知其他相关人员，则会发生许多不该发生的错误。

传统的程序设计技术忽略了数据和操作之间的内在联系，这种方式所设计出来的软件系统内部结构与问题域并不一致，令人难于理解。实际上，用计算机解决的问题都是现实世

① Roger Pressman. 软件工程——实践者的研究方法. 第6版. 北京：机械工业出版社，2009

界中的问题，这些问题无非由一些相互间存在一定联系的事物组成。每个具体的事物都具有行为和属性两方面的特征。因此，把描述事物静态属性的数据结构和表示事物动态行为的操作封装在一起构成一个整体，才能完整、自然地表示客观世界中的实体。

面向对象的软件技术以对象(Object)为核心，用这种技术开发出的软件系统由对象组成。对象是对现实世界实体的正确抽象，它是由描述内部状态表示静态属性的数据，以及可以对这些数据施加的操作(表示对象的动态行为)，封装在一起所构成的统一体。对象之间通过传递消息互相联系，以模拟现实世界中不同事物彼此之间的联系。面向对象的基本思想[①]主要有：

(1) 现实世界中的事物都是对象，对象间存在一定的关系。

(2) 用对象的属性(Attribute)描述事特的静态特征；用对象的操作(Operation)描述事物的行为特征。

(3) 对象的属性和操作成为一个独立的、不可分的实体，实体对外屏蔽其内部细节。

(4) 通过抽象对事物进行分类。类是具有相同属性和相同操作的对象的抽象描述。每个对象是类的一个实例。

(5) 复杂的对象由简单的对象构成。

(6) 运用抽象原则，可以得到较一般的类和较特殊的类。特殊类继承一般类的属性和操作。

(7) 对象之间通过消息进行通信，实现对象间的动态联系。

考虑设计一所大学信息管理系统。如果采用面向过程的方法，首先将定义存储数据的表结构，包括学生、教授、教室及课程的信息，然后设计出处理数据的模块，登记学生选课情况、安排教授授课、分配授课教室等。模块根据业务需求维护数据库。

如果以面向对象观点分析大学信息系统，将分析现实大学中有哪些对象，以及这些对象之间的相互关系。学生、教授、教室及课程这些都是一个个对象，每个对象都包括自己的一些信息，同时能完成一些事情，如学生包括姓名、学号、专业等信息的同时，还能够做一些事：选择某门课程，交学费等。教授也同样包括自己的一些信息(所授的课程及个人信息等)，同样也能做一些事(输入课程分数及提出教学进度等)。从系统角度出发，教室本身也包含着自己的一些信息(位置和容量等)，也能够做一些事(如告知什么时候空闲及什么时候可以预定)。课程也是这样，包括自己的名称、描述，谁选了这门课，也能完成一些事(通知学生选课等)。传统开发方法与面向对象开发方法得到的软件系统结构如图 11-1 所示[②]。

因此，面向对象与面向过程方法分析问题的思路完全两样，是一种新的思考问题的方式。

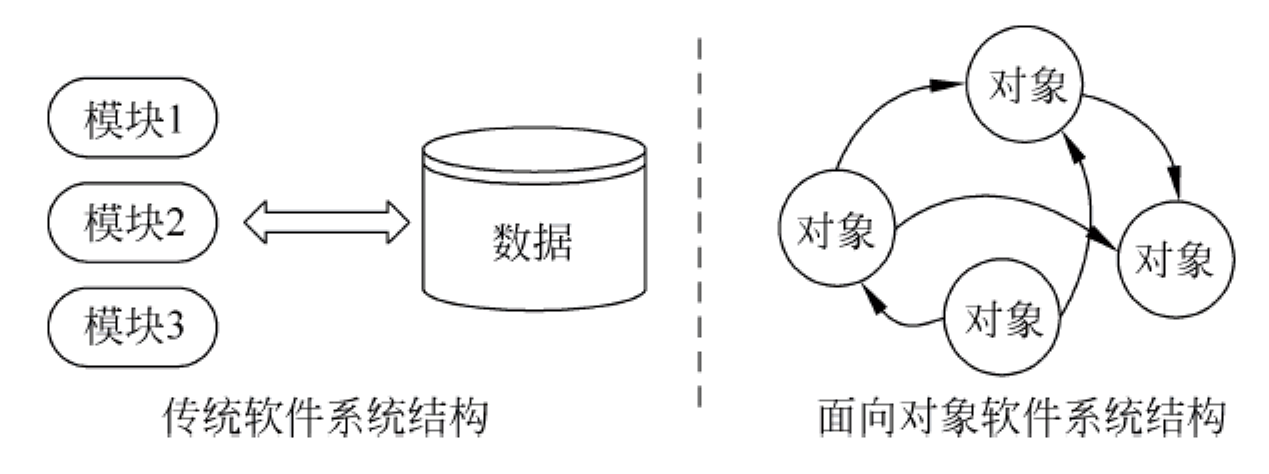


图 11-1 传统过程与面向对象方法

① 麻志毅. 面向对象分析与设计. 北京：机械工业出版社，2008

② Scott W. Ambler. 面向对象软件开发教程. 车皓阳等译. 北京：机械工出版社，2003

11.2　面向对象软件开发方法的优缺点

与传统软件开发方法相比，面向对象开发方法主要有以下这些优点①。

1. 面向对象改变了软件开发的思维方式

如前所述，传统的软件开发方法使设计出来的软件系统与现实世界问题空间并不一致，面向对象方法则如实反映了问题域中的事物，无论是系统的构成部分，还是通过这些成分之间的关系而体现的系统结构，都可直接映射到问题域。面向对象的开发方法与传统的面向过程的方法有本质不同，这种方法的基本原理是：使用现实世界的概念抽象地思考问题，从而自然地解决问题。它强调模拟现实世界中的概念而不强调算法，鼓励开发者在软件开发的绝大部分过程中都用应用领域的概念去思考。在面向对象的开发方法中，计算机的观点是不重要的，现实世界的模型才是最重要的。面向对象的软件开发过程从始至终都围绕着建立问题领域的对象模型来进行：对问题领域进行自然的分解，确定需要使用的对象和类，建立适当的类等级，在对象之间传递消息实现必要的联系，从而按照人们习惯的思维方式建立起问题领域的软件系统模型，模拟客观世界。

2. 利用面向对象技术开发出的软件稳定性高

面向对象方法基于构造问题领域的对象模型，以对象为中心构造软件系统。它的基本做法是用对象模拟问题领域中的实体，以对象间的联系刻画实体间的联系。

因为面向对象的软件系统的结构是根据问题领域的模型建立起来的，而不是基于对系统应完成的功能的分解，所以，当对系统的功能需求变化时并不会引起软件结构的整体变化，往往仅需要做一些局部性的修改。例如，从已有类派生出一些新的子类以实现功能扩充或修改，增加或删除某些对象等。总之，由于现实世界中的实体是相对稳定的，因此，以对象为中心构成的软件系统也是比较稳定的。

3. 软件内部类的可重用性好

对象是一个小的独立封闭体，通过接口与其他对象打交道，这样就容易被重复利用或替换。用已有的零部件装配新的产品，是典型的重用技术，重用是提高生产效率的一个重要方法。面向对象的软件技术在利用可重用的软件成分构造新的软件系统时，体现出较大的灵活性。它可利用两种方法重复使用一个类：一种方法是创建该类的实例，从而直接使用它；另一种方法是从它派生出一个满足当前需要的新类。继承性机制使子类不仅可以重用其父类的数据结构和程序代码，而且可以在父类代码的基础上方便地修改和扩充，这种修改并不影响对原有类的使用。由于使用了面向对象技术的系统，可以像使用集成电路(IC)构造计算机硬件那样，比较方便地重用对象类，或者生产新类替换旧类，来构造软件系统，因此，有人把类称为“软件 IC”。

面向对象的软件技术所实现的可重用性是自然和准确的，在软件重用技术中它是最成

① 麻志毅. 面向对象分析与设计. 北京：机械工业出版社，2008

功的一个。

4. 系统内部容易扩展与维护

由于技术发展、业务竞争、制度改变等因素，现实世界的需求总是不断变化的，这就要求系统有适当的弹性。一个设计良好的面向对象系统是易于扩充和修改的，因此能够适应不断增加的新需求。在面向对象系统中，通过组合不同的对象，扩展出新功能非常容易，同时也可以利用继承或多态设计出新对象，替代旧对象，使系统具备新特性。面向对象的软件技术符合人们习惯的思维方式，因此用这种方法所建立的软件系统容易被维护人员理解。类把易变的数据结构和部分算法封装起来，构成独立性很强的模块，他们可以主要围绕派生类来进行修改、调试工作。通过向类的实例发消息即可调用它，观察它是否能正确地完成要求它做的工作。由于类是个独立体，对类的测试通常比较容易实现，如果发现错误也往往集中在类的内部，比较容易修改与调试。

总之，面向对象的许多优点是长期的，当需要扩展和完善软件系统时，面向对象的优点才会显现出来。即使使用了面向对象技术，也不能保证能构建出正确的系统。面向对象技术只是增加了项目成功的可能性，项目成功与否依靠的是所有项目参与人员对面向对象思想的理解程度。

应用了面向对象技术并不会减少开发时间，相反，初次使用这种技术开发软件，可能比用传统方法所需时间还稍微长一点。因为，面向对象要求开发人员需要更关注需求、分析和设计，必须花很大精力去分析系统中有哪些对象，每个对象应该承担什么责任，所有这些对象怎样很好地合作以完成预定的目标。这样做换来的好处是：提高了目标系统的可重用性，减少了生命周期后续阶段的工作量和可能犯的错误，提高了软件的可维护性。

11.3 面向对象的基本概念

使用面向对象思想进行软件开发，要先弄清楚面向对象的一些概念。简单来看面向对象的概念并不难理解，但是真正掌握这些概念，并能运用面向对象技术开发软件系统还是需要一定的时间的。

11.3.1 对象

对象(Object)是人们要进行研究的任何事物，从最简单的整数到复杂的飞机等均可看做对象，它不仅能表示有形的具体的事物，如一辆汽车，还能表示抽象的无形的，如规则、概念、计划或事件。对象是现实世界中某个实际存在的事物，它可以是有形的(如一辆汽车)，也可以是无形的(如一项计划)。一个对象就是一个独立存在的客观事物，构成世界的一个独立单位，它由一组属性和基于属性的一组操作构成。每个对象都有名称，也称为对象标识。属性和操作是对象的两大要素。属性是对象静态特征的描述，操作是对象动态特征的描述。如家里的宠物是一只叫小白的狗，白毛、体型矮小是它的静态特征，会汪汪叫，撒娇是它的动态特征(图 11-2)。

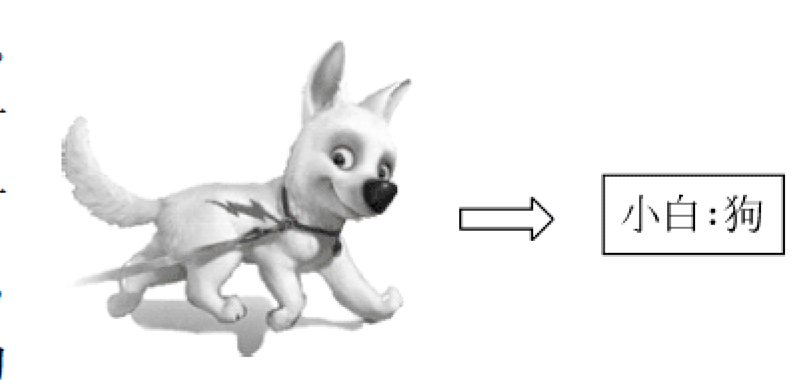


图 11-2 对象与狗

11.3.2　类

类(Class)表示一组相似的对象。将具有相同或相似性质的对象进行抽象就得到类，它为属于该类的全部对象提供了统一的抽象描述，其内部包括属性和方法两个部分。对象的抽象是类，类的具体化就是对象，也可以说对象是类的实例。

图 11-3 给出了人们对客观事物抽象的过程。例如，客观世界中的每一匹马都属于动物类，其中具体的一匹马就是动物类的一个实例，即一个动物对象。在面向对象程序设计里，类是创建对象的模板，是同种对象的集合与抽象。它包含所创建对象的属性描述和行为特征的定义。在面向对象程序设计中，总是先定义类，再用类生成其对象。

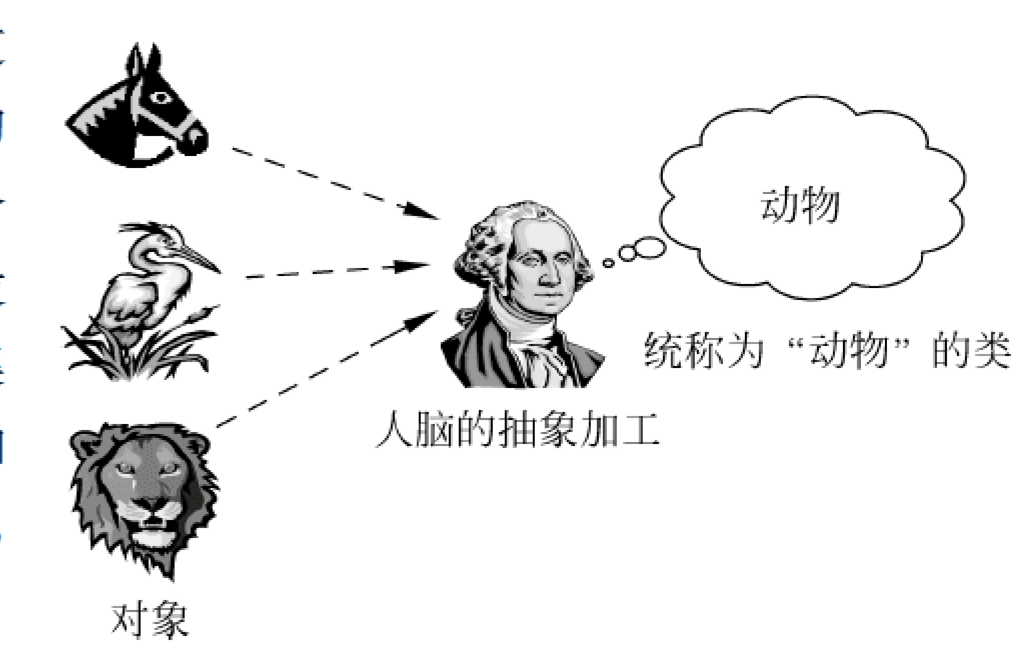


图 11-3　类来源于人脑对事物的抽象

又例如，在一个学生管理系统中，“学生”是一个类。“学生”类具有“姓名”、“性别”、“学号”和“年龄”等属性，还具有“注册”和“选课”等方法。某个具体的学生“Jack”就是“学生”类的实例。

11.3.3　属性和方法

属性(Attribute)是对象知道的信息。在确定问题域中的对象时，要弄清楚对象所知道的信息。现实世界中对象本身有非常多的信息，但在研究的问题域中，对系统有意义的属性只有特定的几个。例如，学生“Jack”对象，有“学号”、“姓名”、“年龄”、“身高”和“头发颜色”等属性，但在学籍管理系统中，学生对象只需要知道“学号”、“姓名”、“学分”等属性即可(图 11-4)。因此，在确定对象属性时，应选择那些“属于”对象而且满足问题域的需要的属性集。

学生
学号 姓名 学分 学籍状态
查询() 计算学分() 修改学籍状态()

图 11-4　对象的属性和操作

方法(Method)是对象有能力完成的事情，定义了对象的行为。总体来说方法可以分为 3 类：①以某种方式操作对象的属性，如增加、删除、修改和查询学生的学号或姓名；②完成计算的操作，如计算学生已取得的总学分；③改变对象状态的操作，例如，根据学生所取得的学分修改学籍状态为毕业、退学或是复读，操作一般都是围绕对象的属性进行的。

11.3.4　抽象、封装和信息隐藏

抽象(Abstraction)就是忽略事物的非本质特征，只注意那些与当前目标有关的本质特征，从而找出事物的共性。在面向对象开发方法中，对象是对现实世界中的事物的抽象，类是对对象的抽象，一般类是对特殊类的抽象。抽象是分析过程，帮助人们忽略与问题域无关的信息，描述系统感兴趣的事。换句话说，抽象出系统感兴趣的类、属性和方法，而

将其他因素忽略。例如，在学校里，学生 Jack 考虑的主要是他的“学号”、“姓名”、“成绩”、“学分”等与教学有关的信息，而如果 Jack 到了银行，考虑的则是“账户”、“余额”、“交易记录”等信息。

信息隐藏是为了让程序容易维护，限制外部对对象的内部信息（属性）的访问以及隐藏的对象方法实现细节。信息隐藏的基本原则就是，如果一个对象想获取另一个对象的信息，必须先征得同意，而不能直接拿来就用。这个做法很容易理解，如想知道某个人的姓名，应该直接询问这个人，而不是抢过他的身份证直接查看。

封装（Encapsulation）是按照信息隐蔽的原则，把对象的属性和方法结合成一个独立的系统单位，并尽可能隐蔽对象的内部细节。通过封装，使外部的对象只能使用对象提供的接口访问对象的属性。由于外部对象不能直接操作对象的属性，从而降低了对象间的耦合度。

汽车是一个封装的典型例子，大量的零部件和操控装置被封装在汽车外壳中并被隐藏起来（图 11-5），提供给人们的只是仪表盘、方向盘和控制手柄，驾驶者无须了解汽车的内部构造和原理，通过方向盘和控制手柄即可操纵汽车。

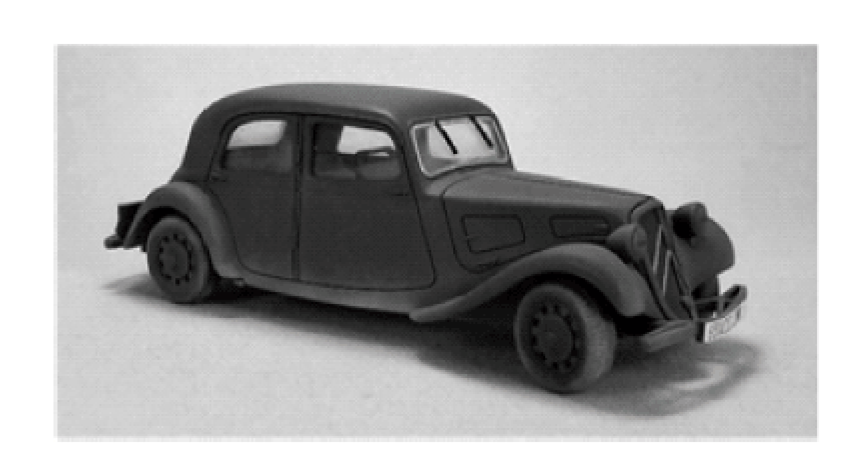



图 11-5　汽车外壳封装了内部的具体部件

通常在实现一个类时，类里的所有属性都设置成私有的，外部的类需要访问时，需要通过方法，而不是直接访问。但严格的封装有时也会带来问题，如编程麻烦，影响执行效率等。

11.3.5　继承

继承（Inheritance）是指特殊类自动地拥有或隐含地复制其一般类的全部属性和操作。这种机制也称做一般类对特殊类的泛化（Generalization）。不同类之间经常会存在相似性，两个以上的类也会经常共享相同的属性或相同的方法。通过利用继承机制建立“is a”或“is like”关系，可以很容易地复用已经存在的数据和代码，避免代码重复编写。继承表达了对象的一般与特殊的关系，特殊类的对象具有一般类的全部属性和方法。

图 11-6 是一个“车”的类继承示例。机动车具有车的全部属性和方法，同样，汽车具有机动车的全部属性和方法。

一般和特殊是相对而言的，在车和机动车之间，车是一般类（基类、超类、父类），机动车是特殊类（子类）；在机动车和汽车之间，机动车是一般类，汽车是特殊类。另外，继承具有传递性，如汽车具有车的全部属性和服务。

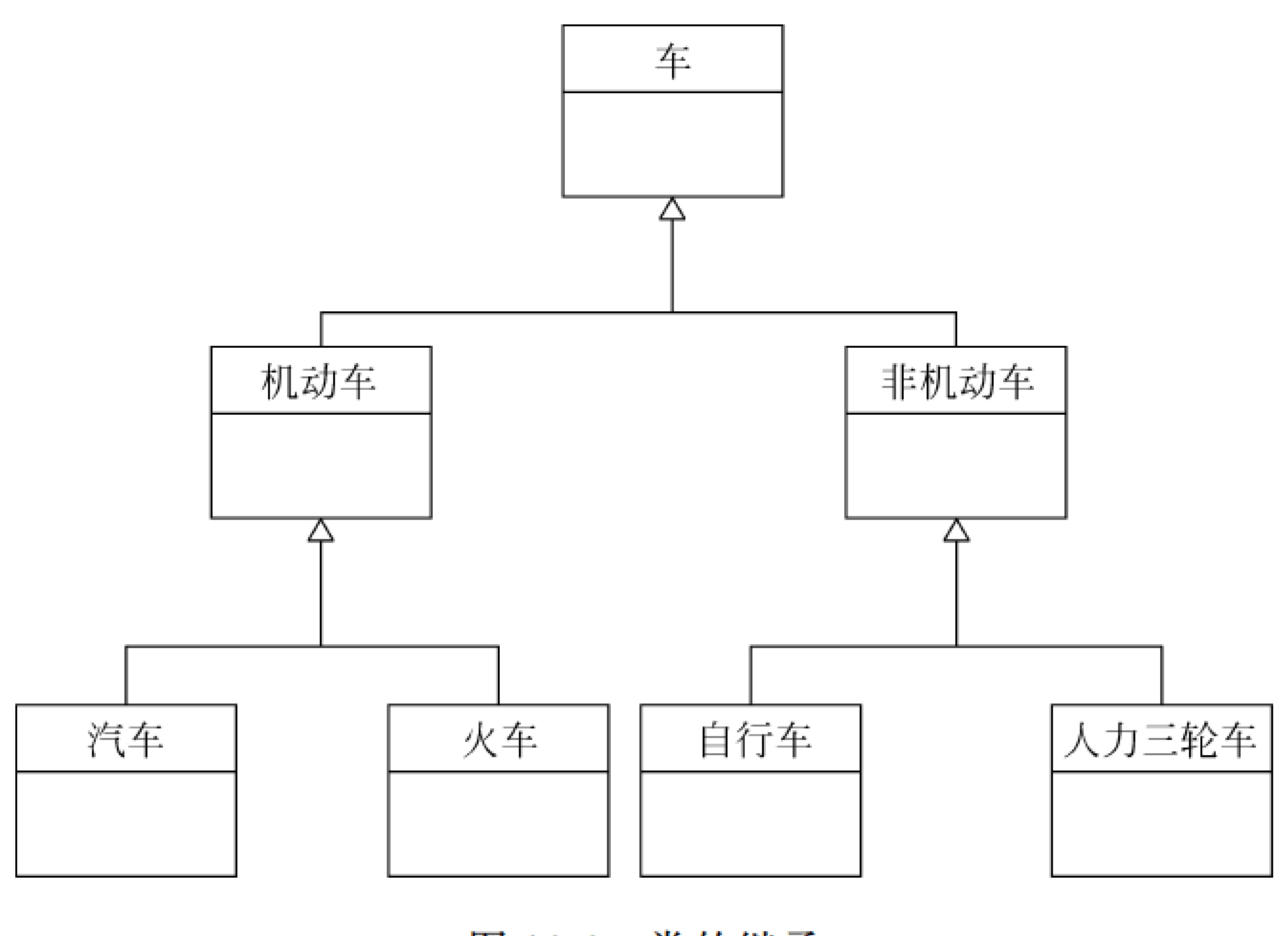


图 11-6　类的继承

11.3.6　多态

多态(Polymorphism)是指在具有继承关系的类层次结构中定义同名的方法或属性,但每个类的属性和方法具有不同的含义,即具有不同的数据类型或表现出不同的行为。也就是说,针对同一个消息时,这些类都可对其响应,但所表现出来的行为却是不同的。

图 11-7 形象有趣地解释了这个概念[①]:当主人向宠物们发出一个“叫”的消息(指令)后,同样的消息被不同的宠物对象接收时,其反应是不同的,鸭子叫的是“嘎嘎”,猫叫的是“喵喵”,而狗是“汪汪”。

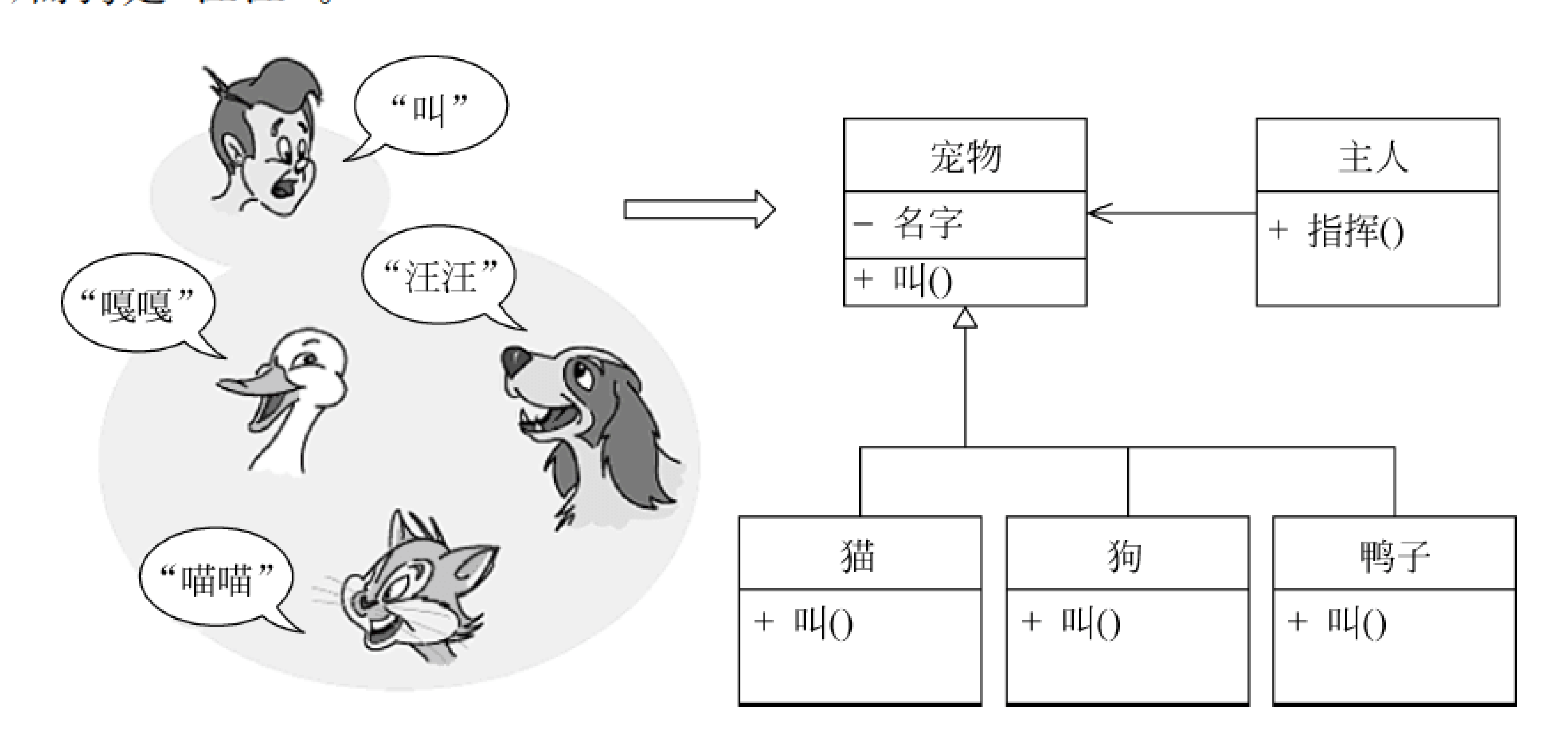


图 11-7　类的多态

多态使对象可以在事前不知道其他对象的类型时就与其协作,如主人在发出“叫”的指令时,不需要知道接受指令的宠物是什么。这样的做法减少了面向对象软件的类之间的耦合,从而降低维护难度。

① 鄂大伟.信息技术基础.第 2 版.福建:厦门大学出版社,2009

11.3.7 关联

关联(Association)是两个或多个类之间的一种静态关系。在现实世界中,对象之间是有联系的,描述对象间的关系非常重要,可以帮助人们定义对象是如何交互的。这种关联描述的是对象间存在静态的结构关系,有了这种关系后,就可以实现动态的交互。如每个"学生"对象拥有一台"计算机",学生与计算机之间是一种静态拥有关系(图 11-8)。学生使用计算机编写程序,这是与计算机的一种动态交互。如果学生与计算机之间不存在"拥有"关系,就不能使用计算机。因此,记录对象间的这些关系,就可以更好地理解对象之间是如何产生交互的。关联在实现时,通过对象的属性值表示。

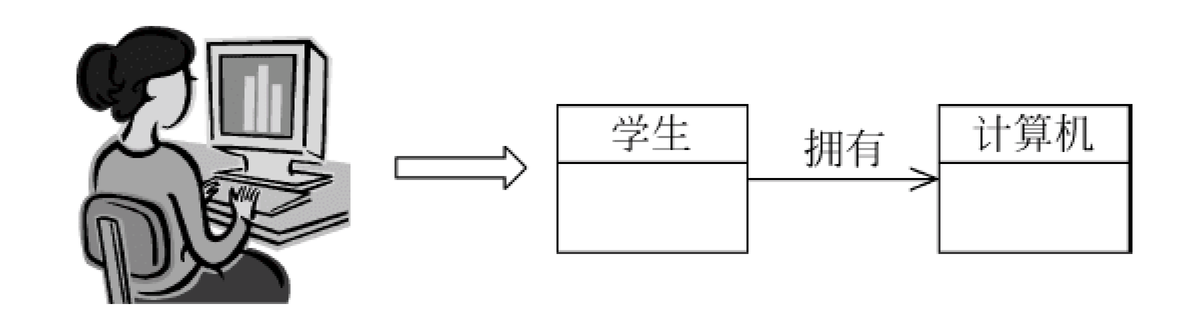


图 11-8 学生拥有计算机

11.3.8 协作

软件系统由对象构成,对象之间通过协作(Collaboration)实现系统的职责。因此,使用面向对象方法建模,寻找类间的协作关系很重要。对象之间的协作是通过相互发送消息实现的。一个对象发送一个操作消息(或请求)给另一个对象,接收消息的对象就执行这个操作。例如,按下电视遥控器的"开机"按钮,遥控器对象向电视机对象发送了一个"开机"消息,电视机对象接收到这个消息后执行"开机"操作。这种发送消息的形式在程序设计里就是一个对象调用另一个对象的方法。因此,发送一条消息至少要包括说明接收消息的对象名、发送给该对象的消息名(即对象名、方法名)。如学生启动编辑器编写代码,而后调用编译器编译源代码,然后执行编译生成的程序,图 11-9 用 UML 的顺序图表示了整个过程。

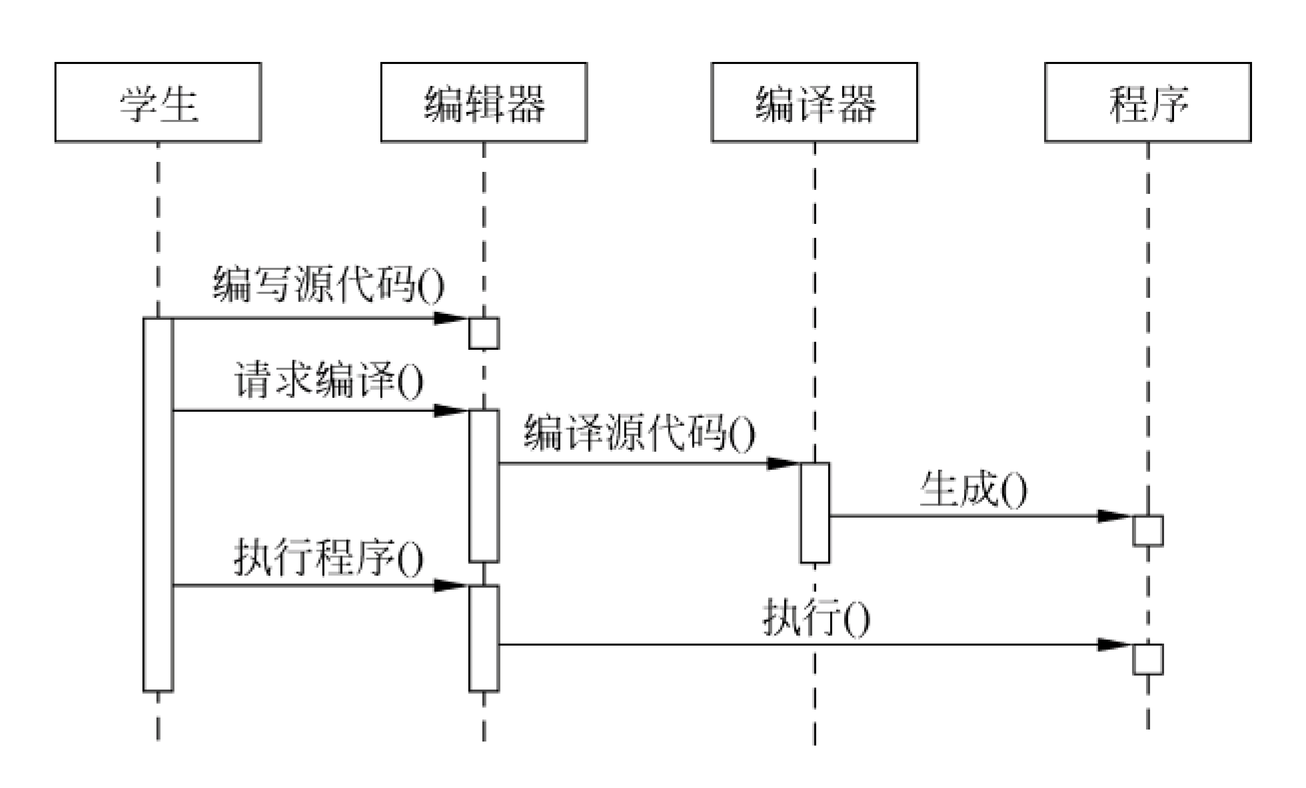


图 11-9 UML 顺序图描述对象协作

11.3.9　聚合

聚合(Aggregation)的概念很容易理解：聚合就是“部分—整体”的关系，是关联的一种类型。如计算机由CPU、内存、主板、显卡和硬盘等组成；一个软件项目团队由小组长、程序员和测试员组成；一个飞机有两个机翼；这些对象由其他对象组合而成，都是聚合的例子，表示它们之间存在“is part of”关系。早期，人们通常把聚合和关联混为一谈，区分和不区分这两个概念的人各执一词而且互不妥协。结果，尽管理由各不相同，许多建模师还是认为聚合是重要的。因此，在面向对象建模里还是包含了聚合的概念，但几乎没有任何语意。正如Jim Rumbaugh所言：“就当它是一种建模安慰剂。”

另一种更强的聚合关系叫做组合(Composition)或称为强聚合。组合关系有3层含义：

(1) 部分对象某一时刻只能从属于一个组成整体对象，如一个打印机在某个时刻只是从属于某台计算机，而程序员在某个时刻有可能从属于多个开发团队；

(2) 部分必须总是属于整体，打印机可以从计算机上卸下，装到另一台计算机上，而树叶总是属于某棵树(图11-10)；

(3) 部分对象的创建和销毁由整体对象负责。

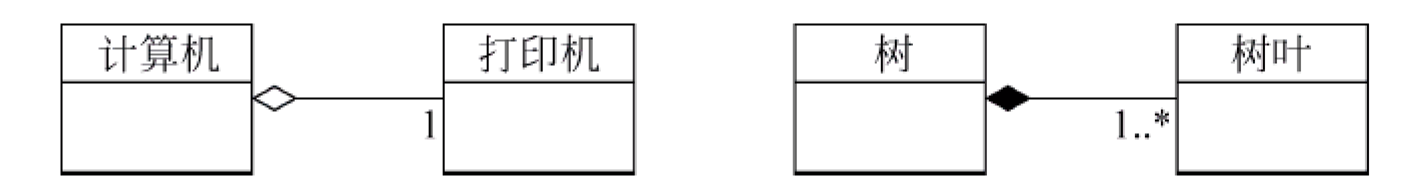


图11-10　面向对象的聚合和组合关系

聚合关系表示事物的整体—部分关系较弱的情况，组合关系表示事物的整体—部分关系较强的情况。在聚合关系中，代表部分事物的可以属于多个聚合对象，为多个聚合对象共享，而且可以随时改变它所从属的聚合对象。代表部分事物的对象与代表聚合事物对象的生存期无关。假如销毁了聚合事物对象，并不一定也要随即删除代表部分事物的对象。而在组合关系中，代表整体事物的对象负责创建和删除代表部分事物的对象，代表部分事物的对象只属于一个组合对象。一旦删除了组合对象，也就随即删除了相应的代表部分事物的对象。

11.3.10　持久性

持久性(Persistence)关注于如何将对象保存到永久存储中，以及如何从永久存储中检索和删除对象。为了使对象持久，必须把对象属性值保存到永久存储中(如关系数据库或文件)，并维护对象之间的相关关系(聚合、继承和关联)。从开发人员的观点来看，软件系统中存在两种类型对象：持久对象和临时对象。例如，超市收银系统的顾客，是一个持久对象，需要把顾客对象存储到某种永久存储中，这样在以后就能使用到。然而像一些临时的界面对象，在程序创建显示后，一旦用户通过它完成处理就可以销毁，而无须保存。

11.4 统一建模语言：UML 概述

11.4.1 UML 的发展历程

面向对象的建模方法始于 20 世纪 80 年代初期，大量有决定意义的思想形成于 20 世纪 90 年代中期，这期间涌现出一些重要方法，包括 Booch、OMT、Shlaer-Mellor、Fusion、OOSE 和 Coad-Yourdon 等。

在 1989—1994 年，面向对象建模语言的数量从不到十种增加到了五十多种。在众多的建模语言中，语言的创造者努力推崇自己的产品，并在实践中不断完善。但是，OO 方法的用户并不了解不同建模语言的优缺点及相互之间的差异，因而很难根据应用特点选择合适的建模语言，于是爆发了一场“方法大战”①。

1994 年 10 月，Grady Booch 和 Jim Rumbaugh 开始致力于这一工作。他们首先将 Booch93 和 OMT-2 统一起来，并于 1995 年 10 月发布了第一个公开版本，称为统一方法 UM 0.8(Unitied Method)。

1995 年秋，OOSE 的创始人 Ivar Jacobson 加盟到这一工作中。经过 Booch、Rumbaugh 和 Jackson 3 人的共同努力，于 1996 年 6 月和 10 月分别发布了两个新的版本，即 UML 0.9 和 UML 0.91，并将 UM 重新命名为 UML(Unified Modeling Language)。

1996 年，一些机构将 UML 作为其商业策略已日趋明显。UML 的开发者得到了来自公众的正面响应，并倡议成立了 UML 成员协会，以完善、加强和促进 UML 的定义工作。当时的成员有 DEC、HP、I-Logix、Itellicorp、IBM、ICON Computing、MCI Systemhouse、Microsoft、Oracle、Rational Software、TI 以及 Unisys 公司。这一机构对 UML 1.0(1997 年 1 月)及 UML 1.1(1997 年 11 月 17 日)的定义和发布起了重要的促进作用。

UML 是一种定义良好、易于表达、功能强大且普遍适用的建模语言。它融入了软件工程领域的新思想、新方法和新技术。它的作用域不限于支持面向对象的分析与设计，还支持从需求分析开始的软件开发的全过程。

面向对象技术和 UML 的发展历程如图 11-11 所示，标准建模语言的出现是其重要成果。UML 符号表示考虑了各种方法的图形表示，删掉了大量易引起混乱的、多余的和极少使用的符号，也添加了一些新符号。因此，在 UML 中汇入了面向对象领域中很多人的思想。

UML 是 Booch、OOSE 和 OMT 方法的结合，同时吸收了其他方法的思想，包括 Wirfs-Brock、Ward、Cunningham、Rubin、Harel、Gamma、Meyer、Odell、Embley、Coad、Yourdon、Shlaer 和 Mellor 等，通过统一这些先进的面向对象思想，UML 成为一种定义明确的、富有表现力的、强大的、可应用于广泛的问题域的建模语言。

---

① UML 官方网站. http://www.omg.org

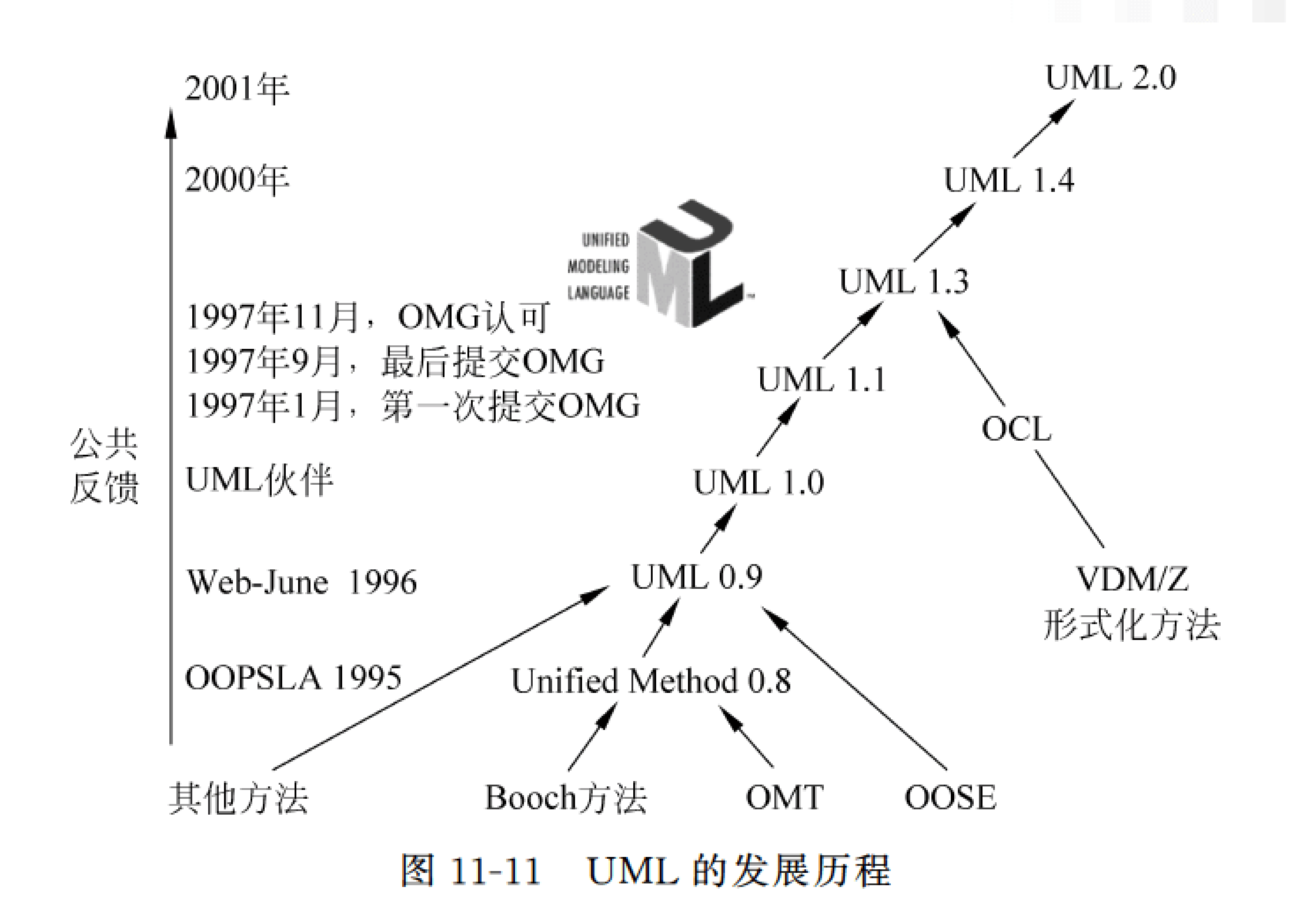


图 11-11　UML 的发展历程

11.4.2　UML 的特点

UML 融合了 Booch、OMT 和 OOSE 方法中的基本概念，而且这些基本概念与其他面向对象技术中的基本概念大多相同，因而，UML 必然成为这些方法以及其他方法的使用者乐于采用的简单一致的建模语言。其次，UML 不仅仅是上述方法的简单汇合，而是在这些方法的基础上广泛征求意见，集众家之长，几经修改而完成的，UML 扩展了现有方法的应用范围。此外，UML 是一种直观化、明确化、构建和文档化软件系统产物的通用可视化建模语言，从企业信息系统到基于 Web 的分布式应用，甚至严格的实时嵌入式系统都适合用 UML 建模。它是一种富有表达力的语言，可以描述开发所需要的各种视图，并以此为基础组建系统。

综上所述，UML 的特点可总结如下。

1. UML 是一种语言

与其他语言一样，UML 提供了用于交流的词汇表及其组词规则，说明如何创建或理解结构良好的模型，但它并没有说明在什么时候创建什么样的模型。

2. UML 是一种可视化的建模语言

软件开发的难点在于项目参与人员之间的沟通和交流，领域专家、软件设计开发人员、客户等各自使用不同的语言交流，对系统的概念模型容易产生错误的理解。另外，阅读程序代码虽然可以推断其含义，但无法正确地理解它，当接手别人的开发工作时，往往由于难以理解而不得不重新实现部分程序。

UML 提供一组具有明确语义的图形符号，可以建立清晰的模型便于交流，同时所有开发人员都可以无歧义地解释这个模型。

3. UML 是一种可用于详细描述的语言

UML 为所有重要的分析、设计和实现决策提供了精确的、无歧义的和完整的描述。

4. UML 是一种构造语言

UML 不是一种可视化的编程语言，它所描述的模型可以映射成不同的编程语言，如

Java、C++和 Visual Basic 等。这种映射可以进行正向工程——从 UML 模型到编程语言的代码生成，也可以进行逆向工程——由编程语言代码重新构造 UML 模型。

5. UML 是一种文档化语言

UML 不是过程，也不是方法，但允许任何一种过程和方法使用它。它可以建立系统体系结构及其详细文档，提供描述需求和用于测试的语言，同时可以对项目计划和发布管理的活动进行建模。

11.5 UML 的视图

UML 是面向对象开发方法的重要工具之一，它是一种能被系统分析员、开发人员或客户所接受的标准设计表示法，类似于电子工程师在电路图设计中所用的标准表示法以及建筑师所画的设计蓝图。UML 是一种可视化的建模语言，能用标准的、易于理解的方式建立系统的设计蓝图，并提供一种机制供不同设计者共享和交流。作为一种建模语言，它使开发人员专注于建立产品的模型和结构，而不是选用什么程序语言和算法实现①。

UML 为系统建模提供了多种不同视图，目的是从不同视角展示一个系统，主要由以下 5 类图组成。

(1) 用例图(Use Case Diagram)

从用户角度描述系统功能，并指出各功能的操作者。

(2) 静态图(Static Diagram)

包括类图、对象图和包图。其中类图描述系统中类的静态结构，不仅定义系统中的类，表示类之间的联系如关联、依赖、聚合等，也包括类的内部结构(类的属性和操作)。类图描述的是一种静态关系，在系统的整个生命周期都是有效的。对象图是类图的实例，几乎使用与类图完全相同的标识。它们的不同点在于对象图显示的是在系统某个时刻的使用场景时类的多个对象实例，而不是实际的类。对象图是类图的一个实例。由于对象存在生命周期，因此对象图只是展示系统某一时刻的对象联系。包图由包或类组成，表示包与包之间的关系。包图有多种用途，最主要用于描述系统的内部结构。

(3) 行为图(Behavior Diagram)

描述系统的动态模型和组成对象间的交互关系。其中状态图描述类的对象所有可能的状态以及事件发生时状态的转移条件。通常，状态图是对类图的补充。在实用上并不需要为所有的类画状态图，仅为那些有多个状态其行为受外界环境的影响并且发生改变的类画状态图。而活动图描述满足用例要求所要进行的活动以及活动间的约束关系，有利于识别并行活动。

(4) 交互图(Interactive Diagram)

描述对象间的交互关系。其中顺序图显示对象之间的动态合作关系，它强调对象之间消息发送的顺序，同时显示对象之间的交互；协作图描述对象间的协作关系，协作图跟顺序

---

① Joseph Schmuller. UML 基础、案例与应用研究. 李虎译. 北京：人民邮电出版社，2004

图相似,显示对象间的动态协作关系。除显示信息传递外,协作图还显示对象以及它们之间的关系。如果强调时间和顺序,则使用顺序图;如果强调上下级关系,则选择协作图,这两种图合称为交互图。

(5) 实现图(Implementation Diagram)

其中构件图描述代码部件的物理结构及各构件之间的依赖关系。一个构件可能是一个资源代码构件、一个二进制构件或一个可执行构件,它包含逻辑类或实现类的有关信息。构件图有助于分析和理解构件之间的相互影响程度。配置图定义系统中软硬件的物理体系结构,它可以显示实际的计算机和设备(用结点表示)以及它们之间的连接关系,也可显示连接的类型及构件之间的依赖性。在结点内部,放置可执行构件和对象以显示结点与可执行软件单元的对应关系。

11.5.1 用例图

用例图描述了待开发系统的功能需求,它将系统看做黑盒,从外部执行者的角度来理解系统。用例模型是驱动后续开发的基础。在用例图中主要元素是用例和参与者。

以一个简单的玩骰子的游戏为例介绍用例图。游戏系统模拟掷两个骰子,如果两个骰子点数和为 7,则游戏者赢,否则为输。图 11-12 描述了这个游戏的用例图。

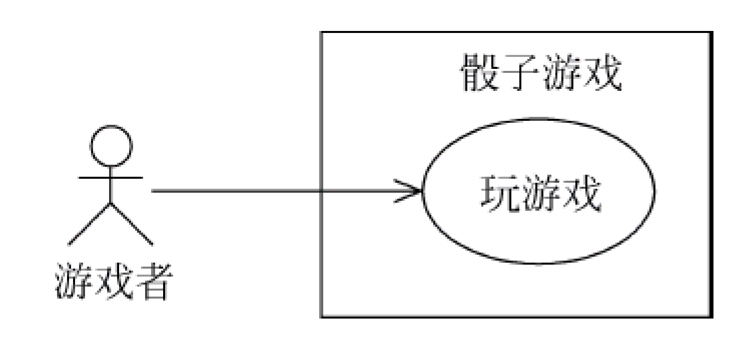


图 11-12 骰子游戏用例图

代表游戏者的直立小人称为参与者(Actor)。椭圆形表示一个用例(Use Case)。参与者触发用例,并与用例进行信息交换,用直线相连。单个参与者可与多个用例联系,反过来,一个用例可与多个参与者联系。对同一个用例而言,不同执行者有着不同的作用:他们可能从用例中取值,也可能参与到用例实现中。矩形框代表系统边界。边界内的用例代表系统需要实现的功能,边界外的参与者代表与系统交互的外部角色。

需要注意的是,尽管执行者在用例图中是用类似人的图形来表示的,但执行者未必是某个人。例如,执行者也可能是一个外界系统,该外界系统可能需要从当前系统中获取信息,与当前系统进行交互。

通常直接列举出用例清单是十分困难的,可以先列出参与者清单,再针对每个参与者列出它的用例,这样分析系统用例就会容易很多。

11.5.2 类图

在面向对象建模技术中,将客观世界的实体映射为对象,并归纳成一个类。类图(Class Diagram)描述系统中类和类之间的静态关系,揭示了系统的静态结构。与数据模型不同,它不仅显示了信息的结构,同时还描述了系统的行为。类图技术是面向对象方法的核心。

类(Class)是对一类具有相同特征的对象的描述。而建立类模型时,应尽量使类与应用领域的概念保持一致,以使模型更符合客观事实,更容易修改、理解和交流。

类描述一类对象的属性(Attribute)和行为(Behavior)。在 UML 中,类的可视化表示为一个划分成 3 个格子的长方形(有时下面两个格子可省略),如图 11-13 所示。一般情况下,由多个单词组合成的类名,每个单词的首字母都要大写,如 DieGame。属性名和操作也

一样,但其首字母不用大写,如 getFaceValue()。

关联(Association)表示两个类之间存在某种语义上的联系。例如,某个人为一家公司工作,一家公司有许多办公室,我们就认为"人"和"公司"、"公司"和"办公室"之间存在某种语义上的联系。在分析设计的类图模型中,则在对应的"人"类和"公司"类、"公司"类和"办公室"类之间建立关联关系。

关联可以有方向,表示该关联单方向被使用。不带箭头的关联可以意味着两个类之间未知、未确定或者该关联是双向关联。多重性表示在这个关联中参与对象的数目的上下界限制。如在骰子游戏中一个"骰子游戏"包含两个"骰子"。

在图 11-13 中,一个骰子游戏(DieGame)里有两个骰子(Die),游戏者(Player)是游戏的玩家,由游戏者发出指令转动骰子,骰子游戏根据结果判断游戏者是输还是赢。这些相互协作、存在信息交互的类之间就有关联。

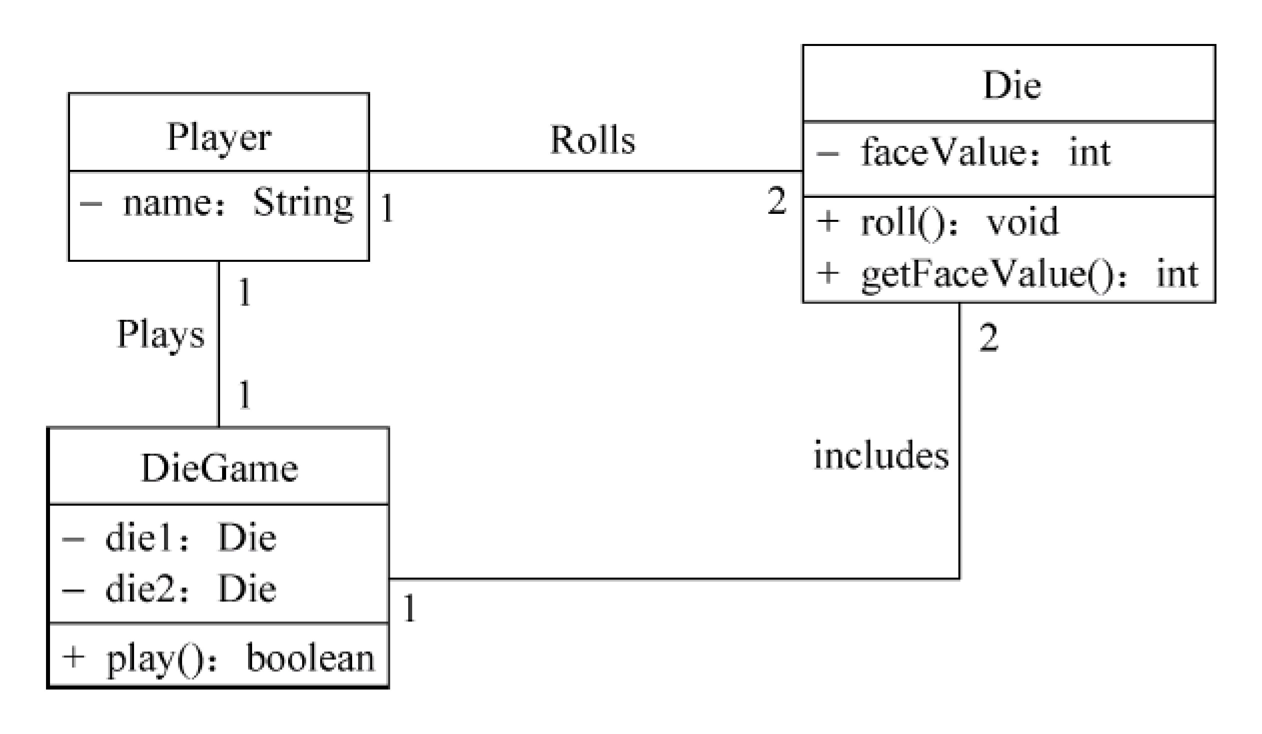


图 11-13 骰子游戏的类图

11.5.3 对象图

对象图(Object Diagram)描述的是参与交互的各个对象在交互过程中某一时刻的状态。对象图可以被看做是类图在某一时刻的实例。

对象(Object)是类的一个实例(Instance),是具有具体属性值的一个具体事物。如游戏者是个叫做"Jack"的人,则"Jack"是"Player"类的一个实例,它的 name 属性值为"Jack"(图 11-14)。

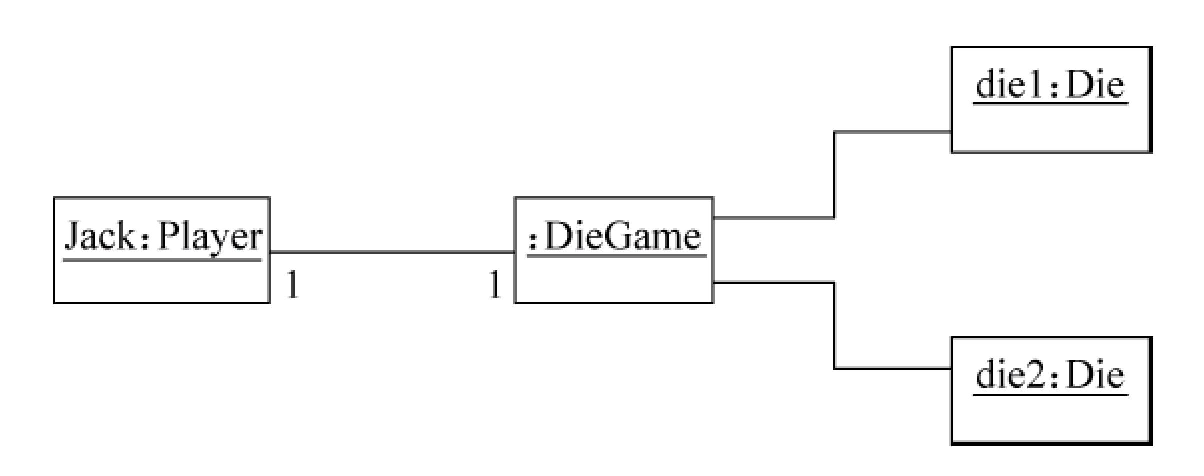


图 11-14 骰子游戏的对象图

对象用矩形表示。但是对象名下面要带下划线。实例名位于冒号的左边,而实例所属的类名位于冒号的右边,如 Jack:Player。实例的名字一般以小写字母开头,有时也可能是一个匿名的对象,则冒号前省略实例名,如:DieGame。

11.5.4 顺序图

顺序图(Sequence Diagram)用来描述对象之间动态的交互关系,着重体现对象间消息传递的时间顺序。

顺序图将交互关系表示为一个二维图。垂直方向是时间轴,时间沿竖线向下延伸。水平方向上列出了在协作中的各独立对象,用一个带有垂直虚线的矩形框表示,并标有对象名和类名。垂直虚线是对象的生命线,当对象实例存在时,角色用一条虚线表示;当对象的过程处于激活状态时,通过在对象生命线上显示一个细长矩形框来表示。

消息用从一个对象的生命线到另一个对象生命线的箭头表示。箭头按时间顺序在图中从上到下排列。当收到消息时,接收对象立即开始执行活动,即对象被激活了。消息还可带条件表达式,表示分支或决定是否发送消息。如果用于表示分支,则每个分支是相互排斥的,即在某一时刻仅可发送分支中的一个消息。

图 11-15 所示的顺序图描述了骰子游戏"玩游戏"用例的实现过程。

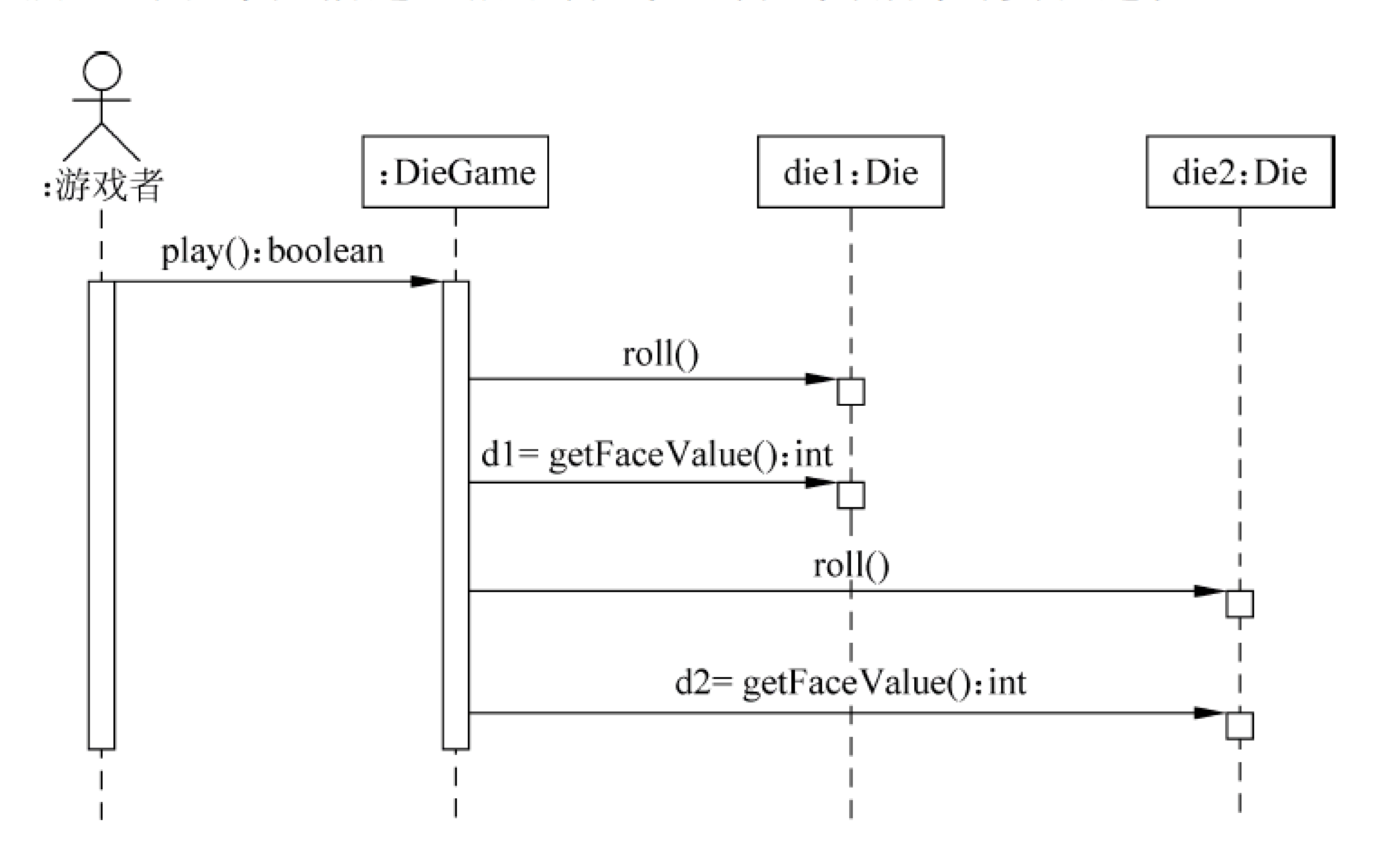


图 11-15 骰子游戏"玩游戏"用例的实现过程

通过顺序图可以知道 DieGame 有个 play 职责(方法),按照顺序图描述的该方法执行的代码片段如图 11-16 所示。

```
public class DiaGame{
    Die die1,die2;
    //…
    public boolean play(){
        die1.roll();
        int d1 = die1.getFaceValue();
        die2.roll();
        int d2 = die2.getFaceValue();
        //…
    }
}
```

图 11-16 DieGame play 职责的代码片段

11.5.5 协作图

协作图(Communication Diagram)用于描述相互合作的对象间的交互关系和链接关系，是交互图的一种，强调对象的作用，而非消息的时间顺序。在 UML 的早期版本中使用 Collaboration Diagram 这个术语，而在 UML 2.0 中使用 Communication Diagram 这个词来替代它，但协作图的本质并没有改变。

虽然顺序图和协作图都用来描述对象间的交互关系，两者在语义上是等价的，也就是这两种图表达的是同一种信息，并且可以将顺序图转换为等价的协作图，反之亦然。顺序图着重体现交互的时间顺序，按照时间顺序布图，而协作图则着重体现交互对象间的静态链接关系，按照空间组织布图。

协作图除了描述对象间的交互外，还表示对象间的各种关系，这些链接关系与类图中的定义相同，在链接的端点位置可以显示对象的角色名等信息。在链接线上，可以用带序号的消息来描述对象间的交互的顺序。消息的箭头指明消息的流动方向。一个消息描述了要发送的消息名、消息的参数、消息的返回值以及消息的序号等信息。

图 11-17 描述了骰子游戏的协作图示例。

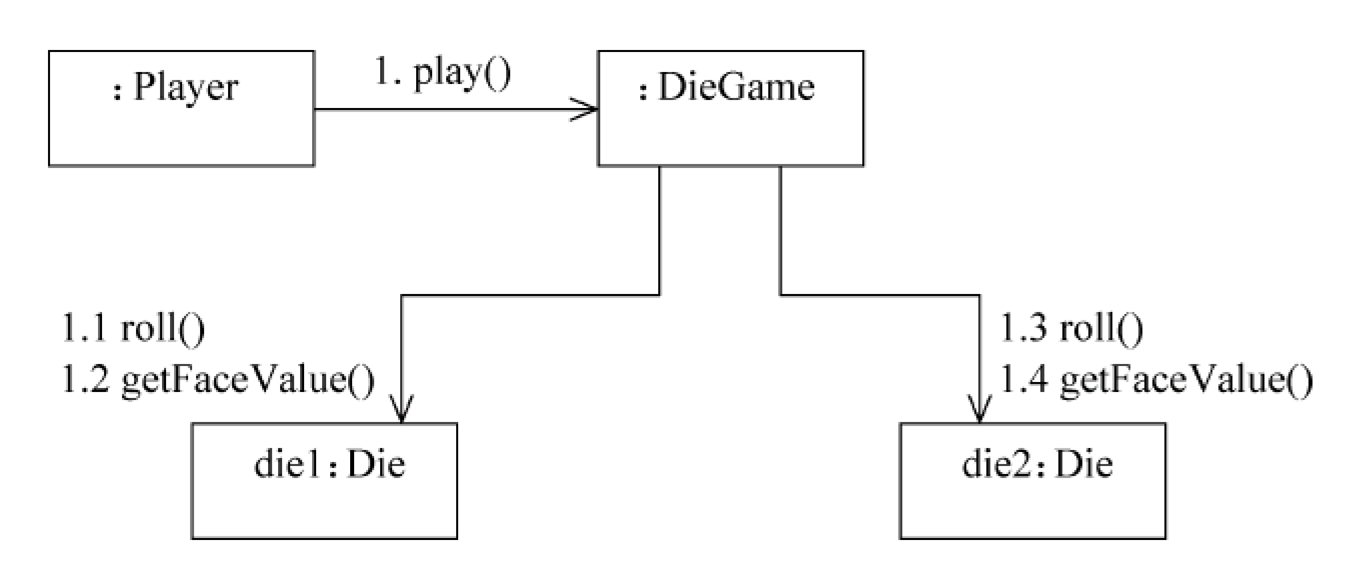


图 11-17　骰子游戏的协作图

11.5.6 状态图

状态图(State Diagram)用来描述一个对象在其生命周期内的所有可能状态及响应外部事件而引起的状态转移行为。状态图是对类所描述对象的补充说明，大多数面向对象技术都用状态图表示单个对象在其生命周期中的行为，它描述了类的所有对象可能具有的状态以及引起状态变化的事件，并不是系统中所有对象都需要进行状态建模，只有当一个对象行为比较复杂，在不同状态下执行不同的行为时才使用状态图为其详细建模。

骰子游戏类图中 DieGame 类是游戏系统的关键对象，对其状态建模如图 11-18 所示。

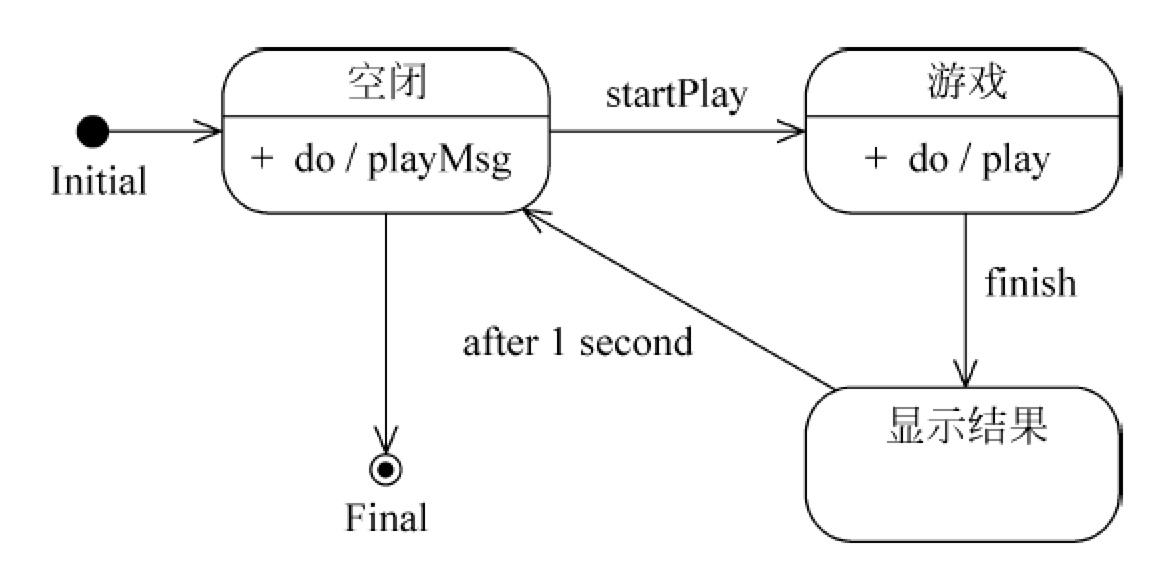


图 11-18　骰子游戏的状态图

11.5.7　包图

包图(Package Diagram)就是对其他 UML 图进行“打包”管理的一种图,打包的目的就是为了按某种方式组织 UML 图,使之更加容易阅读。“包”是 UML 中最常用的管理模型复杂度的机制,也是 UML 中语义最简单的一种模型元素。包就是一种容器,通常用来对一个图的元素(如类或用例)进行分组。UML 包表示比 Java 包或.NET 命名空间更为通用的概念,它表示更为广泛的事物。在包中可以容纳其他任意的模型元素(包括其他的包)。使用包来组织用例可以减少用例太多造成模型复杂的情况,也为后期划分子系统提供了一个参考依据。图 11-19 显示了包图的一个示例,UI 包由 Swing 包和 Web 包组成,依赖于 Domain 包。

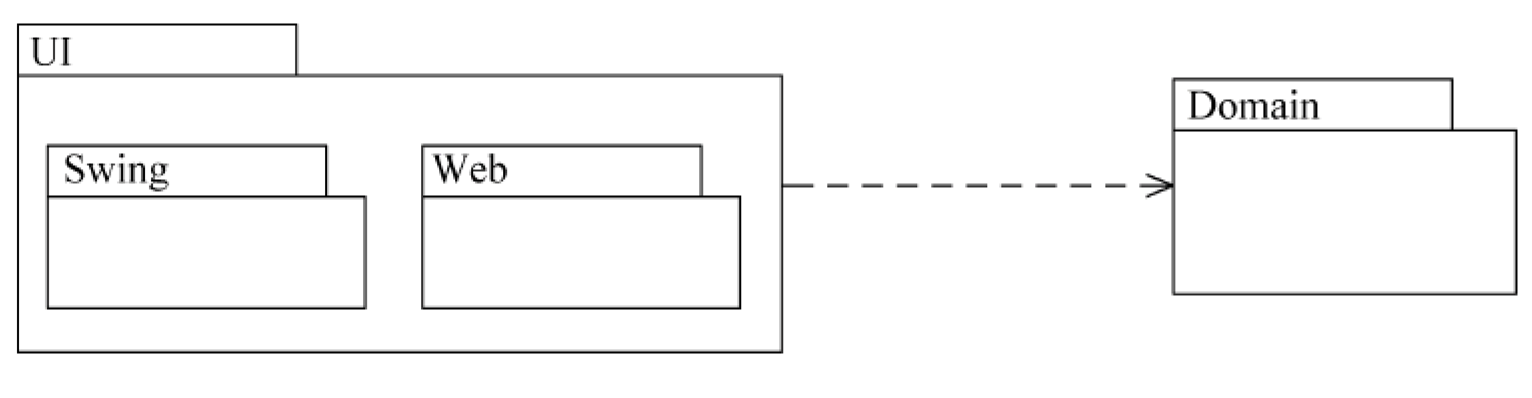


图 11-19　包图

11.5.8　部署图

部署图(Deploy Diagram)展示了系统的构件如何在系统硬件上部署,以及各个构件如何相互连接,同时还表示了软件元素在物理架构上的部署以及物理元素之间的通信方式。UML 2.0 用立方体表示一个结点,连线表示两个结点相连。图 11-20 是一个软件系统的部署图示例。

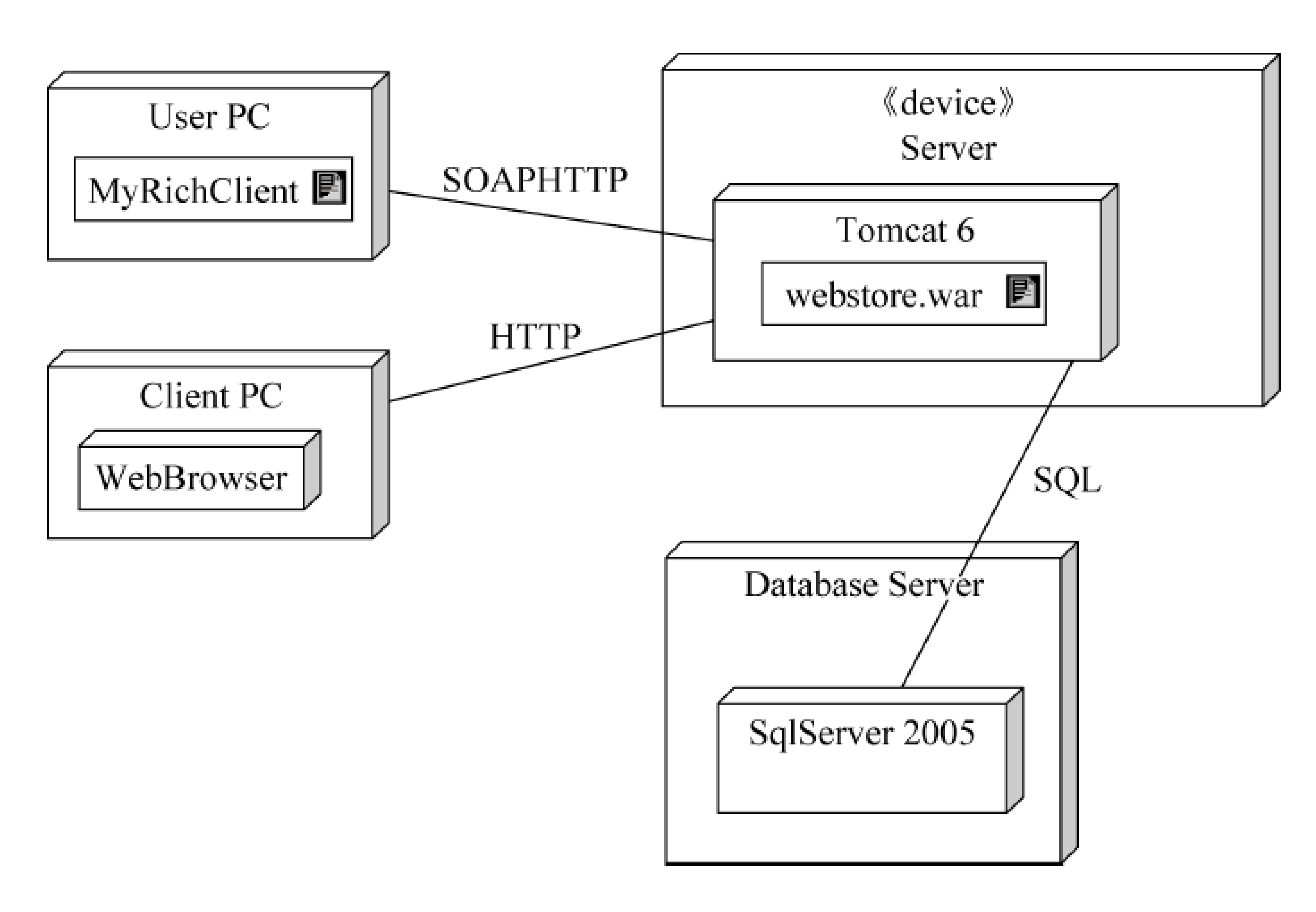


图 11-20　部署图

11.6 面向对象软件的开发过程

面向对象软件开发过程鼓励采用迭代的开发方式，即通过一系列的循环周期演化开发系统，强调开发阶段的无缝集成。通常要往返进行系统的分析与设计，为后续编码提供坚实的基础。

使用面向对象方法，当接触到一个系统时，首先了解当前系统提供什么服务，当前系统的服务如何操作，再分析目标系统必须提供什么服务，目标系统提供的这些服务如何操作，进而考虑如何实现这些服务（这里的实现是逻辑上的，指系统设计；程序设计是指物理上的实现）。总体而言，面向对象软件的开发方法主要分为5个阶段：需求收集、分析、设计、开发和部署，如图11-21所示。下面简述各阶段的内容。

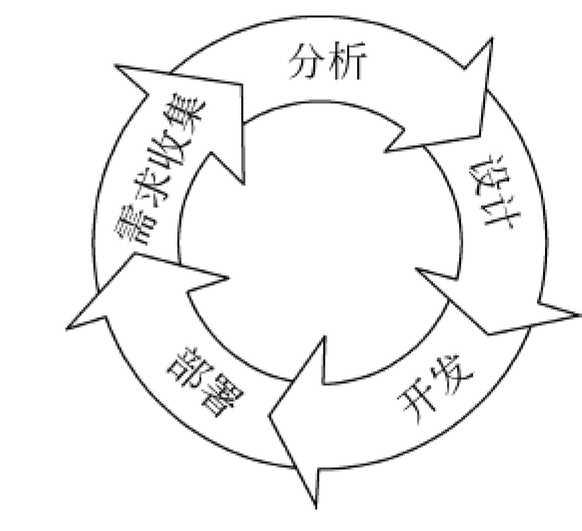


图11-21 面向对象过程框架

在需求收集阶段，首先需要获得对客户业务过程的理解，特别是获得使用目标系统的客户的理解。然后通过使用用例模型来捕获用户需求。通过用例建模，描述对系统感兴趣的外部角色及其对系统（用例）的功能要求。

分析阶段主要关心问题域中的主要概念（如抽象、类和对象等）和机制，需要识别这些类以及它们相互间的关系，并用UML类图来描述。为实现用例，类之间需要协作，这可以用UML动态模型来描述。在分析阶段，只对问题域的对象（现实世界的概念）建模，而不考虑定义软件系统中技术细节的类（如处理用户接口、数据库、通信和并行性等问题的类）。这些技术细节将在设计阶段考虑。

设计阶段利用分析阶段的成果来设计系统的解决方案。设计阶段和分析阶段都可以往返进行直到设计完成。在一些方法学中，分析和设计被当做一个阶段。在这一阶段分析员调整类图的结构，尽量使类的设计达到可重用性高和可维护性好。在设计阶段，利用交互图检查类的操作是否达到要求，以调整或充实类图。同时，交互图也是作为开发编码的一个重要依据。在这一阶段，根据类图推导出系统的数据模型也是一项很重要的任务。

因此，分析的任务就是搞清楚目标系统必须提供什么服务，进而分析这些服务是如何提供的，由哪些对象协作完成。设计的任务主要是把分析得到的结果进一步具体化、规范化整理，明确各对象细节，为构造阶段提供更详细的规格说明，以便能够被面向对象编程实现直接接受。有时，分析与设计的边界是模糊的，并不需要明确区分。

开发（构造）是一个独立的阶段，其任务是用面向对象编程语言将来自设计阶段的类转换成实际的代码。在用UML建立分析和设计模型时，应尽量避免考虑把模型转换成某种特定的编程语言。因为在早期阶段，模型仅仅是理解和分析系统结构的工具，过早考虑编码问题不利于建立简单正确的模型。实现需要充分理解分析与设计的意图，把面向对象设计模型转变成面向对象源代码，在物理上实现系统。

UML模型还可作为测试阶段的依据。系统通常需要经过单元测试、集成测试、系统测试和验收测试。不同的测试小组使用不同的UML图作为测试依据：单元测试使用类图和类规格说明；集成测试使用构件图和协作图；系统测试使用用例图来验证系统的行为；验

收测试由用户进行，以验证系统测试的结果是否满足在分析阶段确定的需求。

当开发完成后，系统就要部署到适当的硬件上运行并要与协同系统集成起来。最后，需要对部署的系统测试，验证是否达到预期效果，备份和恢复机制是否能起作用等。

面向对象开发软件的过程是从一般到具体——从不精确到精确。它开始于对领域的概念理解，然后是系统的高层功能，接着继承深入每个用例、细化模型，最后设计、开发和部署系统。它强调的是在前期尽可能多花些时间对系统进行分析和设计，尽量让编码工作平稳地进行。

11.7　UML 建模工具

自从 1997 年正式发布 UML 以后，众多的 UML 建模 CASE 工具陆续登场，各个建模都有各自的优缺点，其中以 IBM Rational 的 Rational Rose 为代表，为分析员提供了众多的选择。在 UMLChina 网站[①]上列出了目前知名的可供选择的 UML 建模工具。很多 UML 工具是商用的，如 Rose，价格不菲；也有很多免费开源的。选择适应业务和软件应用程序开发需求的 UML 建模工具，要考虑 CASE 工具在 UML 建模能力、项目生命周期支持、双向工程、数据建模、性能、价格、可支持性以及易使用性等方面的不同。

11.7.1　商用建模工具

Rational Rose 是一种基于 UML 的建模工具。在面向对象应用程序开发领域，Rational Rose 是影响其发展的一个重要因素。Rational Rose 自推出以来就受到了业界的瞩目，并一直引领着可视化建模工具的发展。越来越多的软件公司和开发团队开始或者已经采用 Rational Rose，用于大型项目开发的分析、建模与设计等方面。从使用的角度分析，Rational Rose 易于使用，支持使用多种构件和多种语言的复杂系统建模；利用双向工程技术可以实现迭代式开发；团队管理特性支持大型、复杂的项目和大型而且通常队员分散在各个不同地方的开发团队。同时，Rational Rose 与微软公司的 Visual Studio 系列工具中 GUI 的完美结合所带来的方便性，使它成为绝大多数开发人员的首选建模工具；Rose 还是市场上第一个提供对基于 UML 的数据建模和 Web 建模支持的工具。此外，Rose 还为其他一些领域提供支持，如用户定制和产品性能改进。Rose 是商用软件，软件费用比较高。使用 Rose 建模，还必须购买 Rational 绑定的一整套产品，如 Requisite Pro，SoDA，Test Manager 等。

Enterprise Architect 是一个全功能的、基于 UML 的 Visual CASE 工具，主要用于设计、编写、构建并管理以目标为导向的软件系统。它支持用户案例、商务流程模式以及动态的图表、分类、界面、协作、结构以及物理模型。此外，它还支持 C++、Java、Visual Basic、Delphi、C＃以及 VB. Net。EA 虽然也不是免费的，但价格比 Rose 便宜许多，在用户友好性和灵活性方面比 Rose 更胜一筹，特别是序列图。

---

① www. umlchina. com，由 Think 创办。

11.7.2 开源 UML 建模工具

现在也有很多开源的 UML 建模工具，相比较而言，开源的建模工具在功能、易用性以及后期的软件升级上有一定不足。

ArgoUML 是一款开源的 UML 建模工具，最新版 0.24 支持所有 UML 1.4 的标准图形（图 11-22）。它可以运行在任何 Java 平台上，它使用 Java 开发实现，并遵守开源的 BSD 协议。

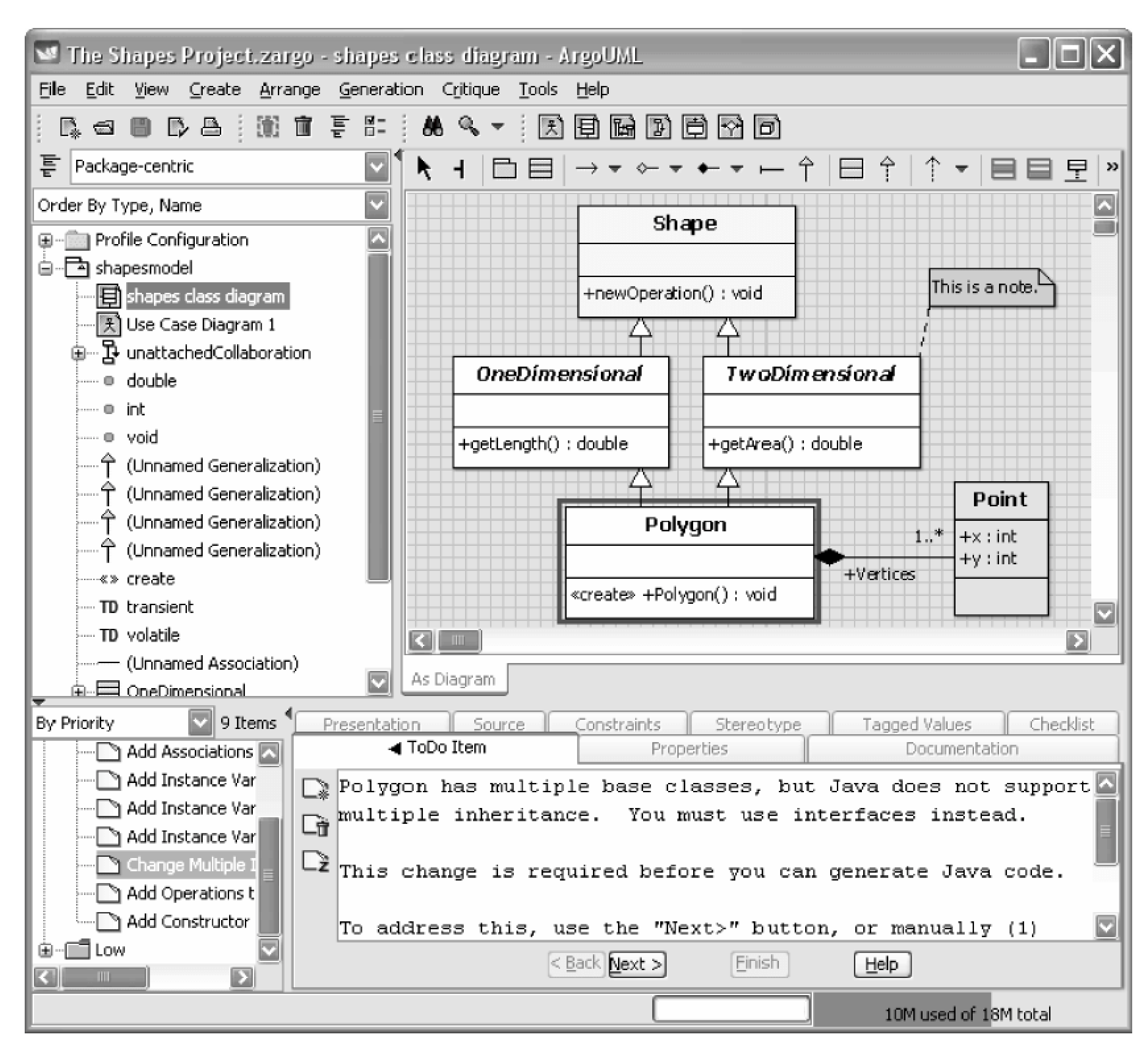


图 11-22 ArgoUML 界面①

本 章 小 结

面向对象思想是人类思考问题的最朴素的思维方式。面向对象与面向过程是完全不同的开发方法，是一种新的思考问题的方式。面向对象方法与传统开发方法相比较，面向对象思想改变了软件开发的思维方式。利用面向对象技术开发出的软件稳定性高，软件内部类的可重用性增强，系统内部容易扩展与维护。

面向对象有 3 个比较重要的概念：将数据和操纵数据的操作封装到单个命名对象中；

① http://argouml.tigris.org/，ArgoUML 官方网站。

继承使类的属性和操作可以被其所有子类和其实例对象继承；多态使得一系列不同的操作具有相同的名字，减少需要用于实现系统的代码行数并方便修改。除此之外，面向对象还有关联、协作、聚合和组合等概念。看起来面向对象的概念并不难理解，但是真正掌握这些概念，运用面向对象技术开发软件系统还是需要花时间才能掌握的。

面向对象开发方法始于20世纪80年代初期，曾涌现出众多的开发方法。到了1994年，经过Booch、Rumbaugh和Jackson 3人的共同努力，推出了UML。UML考虑了各种方法的图形表示，删掉了大量易引起混乱的、多余的和极少使用的符号。

UML是面向对象开发方法的重要工具之一，它是一种能被系统分析员、开发人员或客户所接受的标准设计表示法，类似于电子工程师在电路图设计中所用的标准表示法以及建筑师所画的设计蓝图。UML提供了多种不同视图，以便从不同视角展示一个系统，主要有用例图、类图、对象图、顺序图、协作图、状态图、包图、部署图等。需要注意的是，UML只是一个表示工具，掌握UML并不等于掌握了面向对象方法。

面向对象软件工程鼓励采用迭代的开发方式，即通过一系列的循环周期演化开发系统。总体而言，面向对象开发方法主要分为5个阶段：需求收集、分析、设计、开发、部署。

目前有许多的UML建模CASE工具可供选择，其中以IBM Rational的Rational Rose为代表，各个建模工具都有各自的优缺点。

思考与练习

1. 使用面向对象方法开发的系统与传统方法开发的系统有何本质的不同？面向对象方法有什么优点？

2. 和所有的技术一样，面向对象也存在缺点。请从网上或图书馆查阅资料，总结面向对象的缺点，以及在实施面向对象方法时应该注意的问题。

3. 针对你过去使用的开发方法所构建的系统的不足，总结问题的原因。考虑如果使用面向对象方法，在哪些方面可能会获益。

4. 查阅资料，进一步讨论UML与面向对象方法的关系。

5. 请从网上或图书馆查阅资料，编写小论文介绍Grady Booch、Peter coad、Martin Fowler、Erich Gamma、James Gosling、Brian Henderson-Sellers、Iva Jackson、Bertrand Meyer与James Rumbaugh在面向对象技术领域做出过哪些贡献，为什么这些贡献会那么重要？

6. 封装的目的是什么？在面向对象方法中封装的目的是如何达到的？

7. 请用自己的语言描述继承、多态和聚合，并分别举例说明。

8. 什么时候应采用继承？什么时候不用？提供继承何时适用、何时不适用的例子，并加以讨论。

9. 研究两种不同的面向对象程序设计语言并显示消息传递在语言的语法中是如何实现的，给出示例。

10. 假设要构造一个和用户下棋的系统，哪些种类的UML图对设计该系统有用处？为什么？

11. 仿照本章中提到的骰子游戏的建模方法，为班级提问系统建立模型（用例图、类图、用例实现场景顺序图等）。该系统模拟抽奖过程，由随机种子产生学生学号，让该学生回答问题，回答正确给奖品鼓励，错误不惩罚。

12. 试使用面向对象语言，实现如图 11-23 所示类图描述的类。

Dialer	
−	digits: Vector
−	nDigits: int
+	digit(int): void
#	recordDigit(int): boolear

图 11-23　类图

13. 请依据如图 11-24 所示的顺序图，试着使用面向对象程序设计语言实现 Controller 的 doEvent 方法。

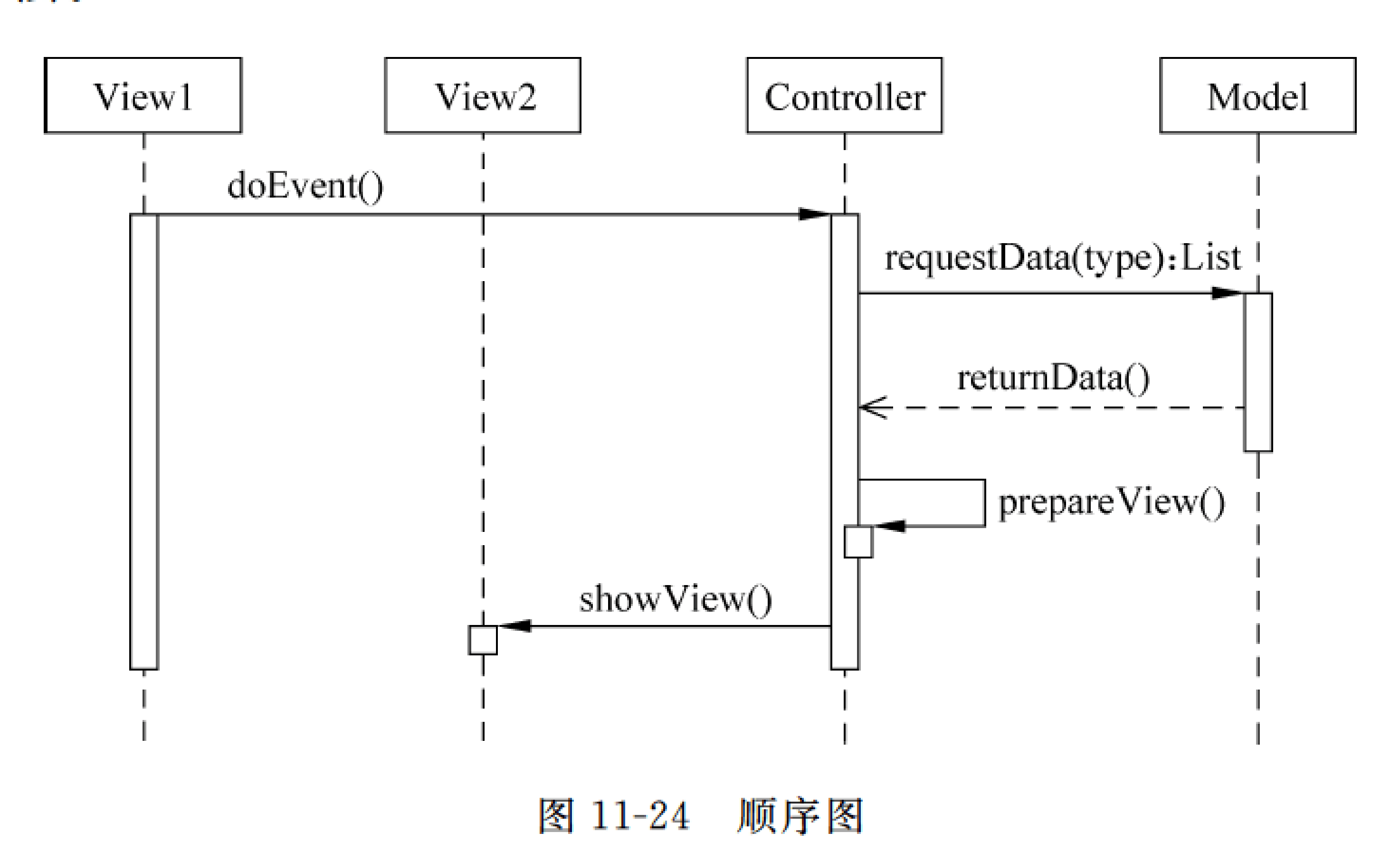


图 11-24　顺序图



第12章 面向对象分析

好的软件的作用是让复杂的东西看起来简单。

——Grady Booch(UML 创始人之一)

面向对象分析(Object-Oriented Analysis,OOA)强调运用面向对象方法,对问题域和系统责任进行分析和理解,找出描述问题域和系统责任所需要的对象,定义对象的属性、操作以及对象之间的关系,建立一个符合问题域、满足用户功能需求的 OOA 模型。

在面向对象分析阶段,对问题域的理解和分析是直接的,对问题域的描述也是采用最直接的方式。在建立的模型中,所采用的概念与问题域中的事物保持最大程度的一致。问题域中有哪些承担职责的事物,在模型中就有相对应的对象,而且对象、对象的属性和操作的命名都强调与客观事物一致。

当前系统分析面临的困难主要有:对问题域和系统责任的理解、人与人之间的交流、需求的不断变化以及软件复用对分析的要求。

OOA 强调从问题域的实际事物以及与系统责任有关的概念出发构造系统模型。这使系统中的对象、对象的分类、对象的内部结构以及对象之间的关系能良好地与问题域中的事物相对应。因此,OOA 有利于对问题域和系统责任的理解。

同时,由于 OOA 充分运用日常生活的思维方法和策略认识和描述问题域,直接使用问题域中的概念和术语,从而改进了各类人员之间的交流。

OOA 遵循封装和信息隐藏的原则,把容易变化的属性及一部分操作封装并隐藏在对象之中,当它们发生改变时,主要影响到的只是对象内部。对象只通过接口对外部产生影响,这样就有效地改变了原来一处修改处处受牵连的情况。以相对稳定的对象作为构建系统的基本单位,减少了因需求改变而造成的系统大变动。

最后,由于对象是一个独立的封装实体,很适合作为一个可复用成分。一组关系密切的类还可以构成粒度更大的可复用成分。满足了软件复用的要求,提高了开发效率。

对于 OOA 主要的活动有以下几个。

(1) 与客户进行沟通,了解用户需求,建立需求模型。

(2) 标识问题域中的对象,分析对象承担的职责。

(3) 分析对象与对象之间的关系。

(4) 定义对象之间的交互。

这些活动需要反复迭代,直至模型完成。

12.1 收集需求

12.1.1 面向对象的软件需求概述

不管是开发什么样的软件，使用什么样的方法，软件开发的第一步都是收集用户的需求。不知道做什么，就不可能成功构建一个系统。传统的软件需求规约基本上采用的是以功能分解的方式来描述系统功能，在这种表述方式中，系统功能被分解到各个系统功能模块中，通过描述细分的系统模块的功能来达到描述整个系统功能的目的(图 12-1)。采用这种方法来描述系统需求，实际上已经包含了部分的设计思想，容易混淆需求和设计的界限。

软件需求规约(SRS)

子系统 1　功能
　　模块 1.1 功能
　　模块 1.2 功能
　　……
子系统 2　功能
　　模块 2.1 功能
　　模块 2.2 功能
　　……

图 12-1　传统过程软件需求说明片段

面向对象软件开发使用用例的方法来描述系统需求。从用户的角度来看，并不想了解系统的内部结构和设计，所关心的是系统所能提供的服务，也就是被开发出来的系统将是如何被使用的，这就是用例方法的基本思想。

面向对象软件开发并不是一开始就考虑系统有哪些对象，而是从对系统将被如何使用的角度开始捕捉需求。如果系统是人机交互的，则考虑人是如何使用这个系统的；如果涉及过程控制的，则考虑机器使用系统的方式；如果系统是协调和控制应用，则考虑其他程序调用系统的方式。总之，把系统放到应用场景中，捕捉系统外部对象如何使用系统，就是面向对象收集需求的基本方式。

用户需求就是用户对所要开发的系统提出的各种要求和期望，其中包括系统的功能、性能等技术性要求，还包括成本、交付时间和资源使用限制等非技术性要求，在此主要讲述功能需求建模。在第 11 章中已经介绍过用例图，使用用例建模是最有效的收集用户需求的技术。用例是重要的需求分析制品，强调了系统的活动视图。用例建模主要针对系统的功能性需求。需要注意的是，用例建模除了绘制用例图外，还需要为每个用例编写用例规约。

用例图对系统的功能以及与系统进行交互的外部事物建模，找出与系统交互的外部事物，并说明它们如何与系统交互(图 12-2)。这样，用户就能够理解未来的系统，开发者也能够正确地理解需求并实现系统。用例图是 OOA 的基础，特别是对人机交互来说，用例是非常重要

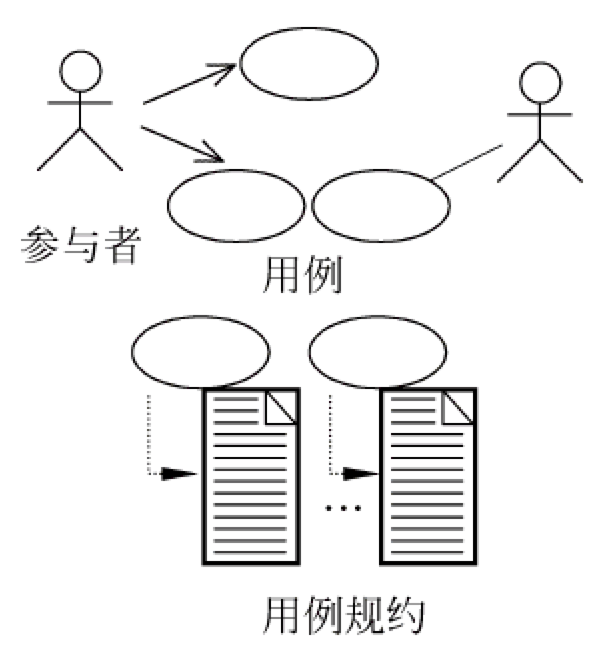


图 12-2　用例模型包括用例图和用例规约

的。整个面向对象开发是以"用例驱动"(Use-Case Driven)的,各种类型的开发活动,包括项目管理、分析设计、测试、实现等都是以系统用例为主要输入,用例模型奠定了整个系统软件开发的基础。

以下的分析还是以第 7 章的 ATM 自动取款机系统为例。

12.1.2 系统范围

在建立需求模型时,首先要确定系统范围①,找出在系统范围以外与系统交互的事物,然后从这些事物与系统进行交互的角度,通过用例来描述这些事物怎样使用系统,以及系统向它们提供什么服务。

没有系统之前,如何描述用户需要什么样的系统?首先把即将要开发实现的软件系统看做一个黑箱,分析这个系统在现实问题域中如何被使用。系统范围指的是一个系统与系统以外各种事物的分界线。在没有分析以前,将开发的系统看成是一个由一条边界包围起来的未知空间,系统只通过边界上的有限个接口与外部的系统使用者(人员、设备或其他系统)进行交互。把边界上的交互情况描述清楚了,也就知道了系统需要提供的服务,从而定义系统的功能需求。图 12-3 描述了系统范围的概念。

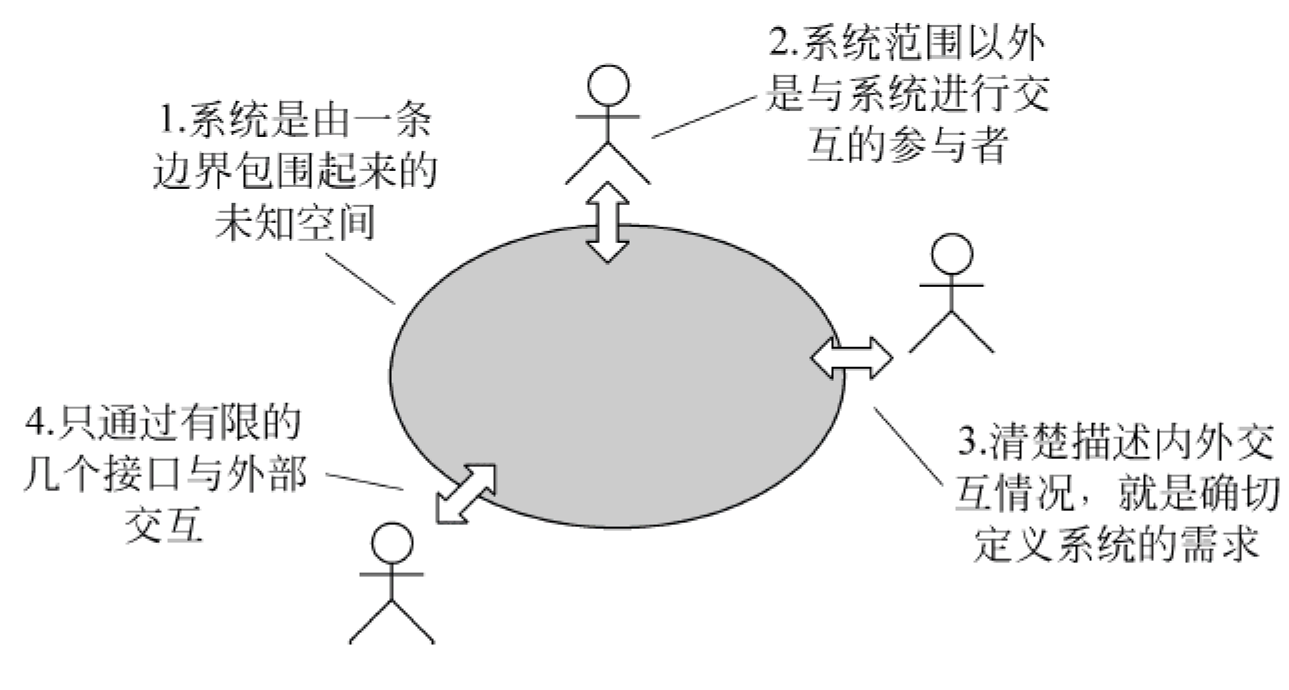


图 12-3 系统范围

认清系统边界,需要分析事物与系统之间的以下几种关系。

(1) 某些事物位于系统边界内,作为系统成分。如超市收银系统中的商品,抽象为系统内的"商品"对象。

(2) 某些事物位于系统边界外,与系统交互,作为参与者。如商品查询系统中的顾客,系统的商品查询没有权限限制,而且无须记录这些顾客信息,为其提供特殊服务。

(3) 某些事物可能既有一个对象作为其抽象描述,而本身(作为现实世界中的事物)又是在系统边界以外与系统进行交互的参与者。如超市中的收银员,他本身是现实中的人,作为参与者;在系统范围内,又有一个相应的"收银员"对象来模拟其行为或管理其信息,作为系统成分。

(4) 某些事物即使属于问题域,也与系统责任没有什么关系。如超市中的保安员,在现实中与超市有关系,但与所开发的系统超市商品管理系统并无关系。这样的事物既不在系

① http://www-900.ibm.com/developerWorks/cn/rational/r-usecase-atm/

统边界内,也不作为系统的参与者。

参与者是由系统的边界决定的,如果所要定义的系统范围仅限于ATM机本身,那么除了使用ATM的银行客户是参与者外,银行的后台服务系统都是与ATM交互的外部系统,可以抽象为一个参与者。图12-4显示了ATM机与外部的参与者。

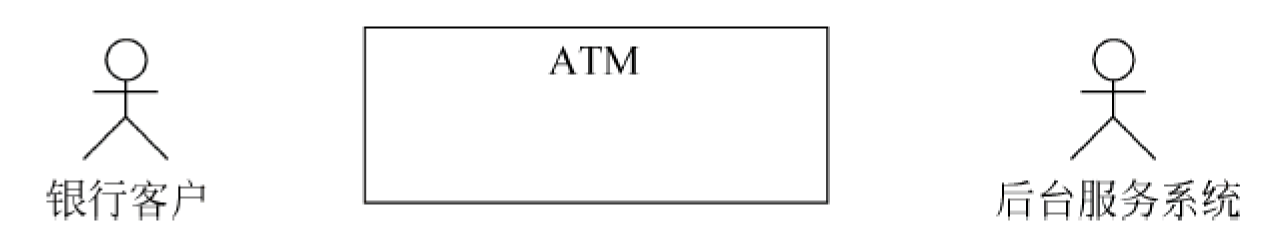


图12-4 ATM机系统范围

如果所要定义的系统边界扩大至整个银行系统,研究的是整个银行系统提供的服务。那么,ATM机和后台服务系统都是整个银行系统的一部分。这时后台服务系统就不再被抽象成为一个参与者(图12-5)。

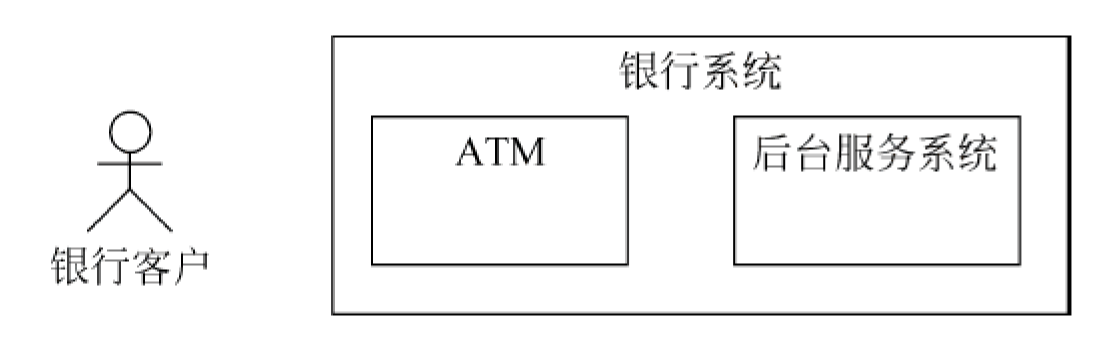


图12-5 银行系统范围

值得注意的是,用例建模时不要将一些系统的组成结构作为参与者来进行抽象,如在ATM系统中,凭条打印机只是系统的一个组成部分,不应将它抽象成一个独立的参与者;在一个MIS管理系统中,数据库系统往往只作为系统的一个组成部分,一般不将其单独抽象成一个参与者。

12.1.3 参与者

分清楚系统边界是为了捕捉外部有哪些事物与系统交互,从而得到系统的参与者。参与者(Actor)表示与系统进行交互的任何人或物,包括人、外部系统、其他设备或外部事件。

1. 人

首先从直接使用系统的人员中发现参与者。这里强调的是对系统的直接使用,而不是间接使用。如银行客户通过柜台取钱,则对于银行内部的业务系统来讲,银行客户不是参与者,因为他是通过银行职员完成业务,银行职员直接使用银行业务系统完成操作,银行职员才是银行业务系统的参与者,如图12-6所示。

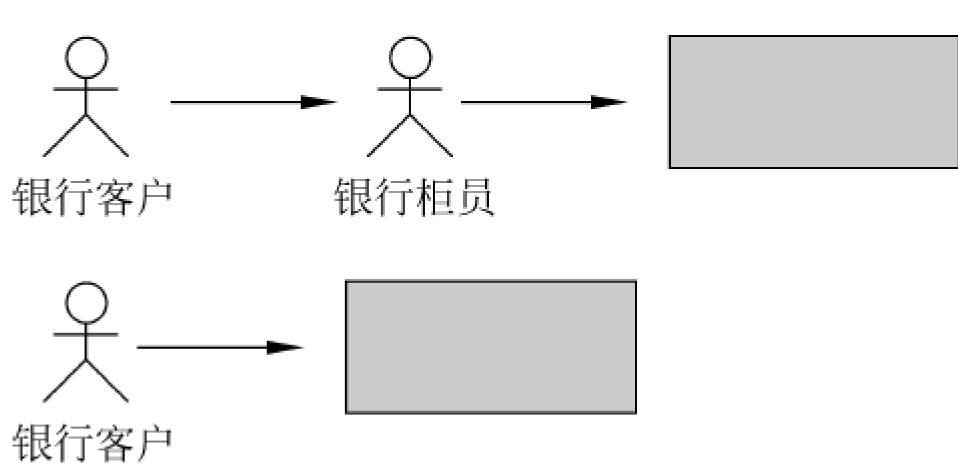


图12-6 系统的直接使用者才是参与者

参与者是为角色建模,而不是为具体的、实际的人。特定的人在系统中可扮演不同的角色。例如,有的系统的操作人员只有一位,他既可能是前台业务数据的输入者,也可能是后台数据的管理员,同一个人扮演了两种不同的角色,反映为两种不同的参与者。

可以在描述参与者时提供一个现实世界的例子，使得角色更形象，更容易理解。

2. 外部系统

所有与系统交互的外部应用系统都应看做是参与者。

从系统边界的角度来说，应该把与软件系统运行在同一平台上的应用系统看做是外部的应用。相对于当前正在开发的系统而言，外部应用系统可以是其他子系统、上级系统或任何与它进行协作的系统，但对它的开发并不是当前系统的开发小组的责任。如超市的收银系统需要银联授权系统协助才能完成收银刷卡支付功能，但是银联授权系统并不需要超市收银系统的开发小组负责开发。

3. 设备

应识别所有与系统交互的设备。这样的设备与系统相连，向系统提供外界信息，或在系统的控制下运行。通常，不包括监视器、键盘、鼠标和其他的标准的用户接口类型设备，考虑外部传感器(输入信息)或受控马达(输出信息)等诸如此类的设备。

4. 外部事件

当构造实时和异步交互的系统时，将外部事件识别为潜在的参与者就变得更加重要了。由外部事件引导，发现系统需要提供的潜在功能对需求分析十分有帮助。

例如，由时间的流逝而激发系统的活动是常见的情况。可以把时间事件作为一个参与者，也可以把时间事件作为系统的一部分，还可以把二者结合起来使用。

对于 ATM 系统，根据上述 4 种分类可以帮助我们找到外部的参与者：银行客户使用 ATM 机提供的服务，银行管理员负责维护和管理 ATM 系统，这是直接使用系统的人员；ATM 机也需要与后台服务系统进行通信以获得有关用户账号的相关信息，这是与系统交互的外部应用系统(图 12-7)。

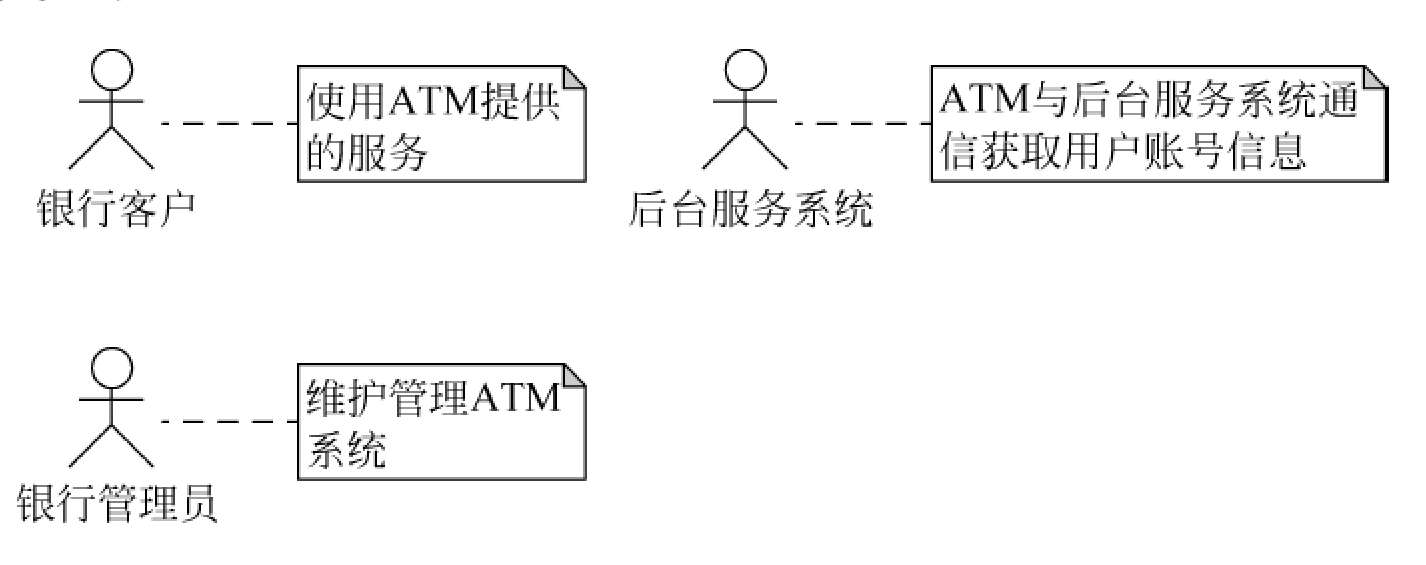


图 12-7 ATM 系统的参与者

有时需要在 ATM 系统内部定时地执行一些操作，如检测系统资源使用情况、定期地生成统计报表等。从表面上看，这些操作并不是由外部的人或系统触发的，而是由异步事件触发。对于这类功能需求，如果想用用例方法表述，可以仿照第 4 类参与者模式，抽象出一个系统时钟或定时器参与者，利用该参与者来触发这一类定时操作。从逻辑上讲，这一参与者应该被理解成是系统外部的，由它来触发系统所提供的用例对话(图 12-8)。

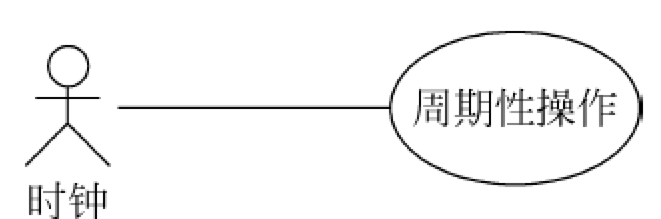


图 12-8 时钟参与者

12.1.4 用例

用例(Use Case)用于表示系统所提供的服务,它定义了系统是如何被参与者所使用的,描述的是参与者为了使用系统所提供的某一完整功能而与系统之间发生的一段对话。

有了参与者之后,可以根据参与者确定系统的用例。主要是看各参与者需要系统提供什么样的服务,或者说参与者是如何使用系统的。寻找用例可以从以下问题入手:

- 在交互过程中,它们是怎样使用系统的服务来完成它们的任务以达到目的的?
- 是什么事件引起与系统进行交互的活动?
- 当外部有事件发生时,该参与者需要通知系统吗?
- 当系统内部有某些事件发生时,需要通知该参与者吗?
- 已定义的系统功能是否满足所有的业务需求?
- 该参与者会在系统中创建、修改、删除或访问任何数据吗?

例如 ATM 中的主要使用者是银行客户,对于客户这个参与者来说主要使用自动提款机来进行银行账户的修改密码、查询、取款和转账交易。而后台服务系统需要在客户使用 ATM 机交易时提供账户查询等辅助作用,是辅助参与者。ATM 系统部分用例图如图 12-9 所示。

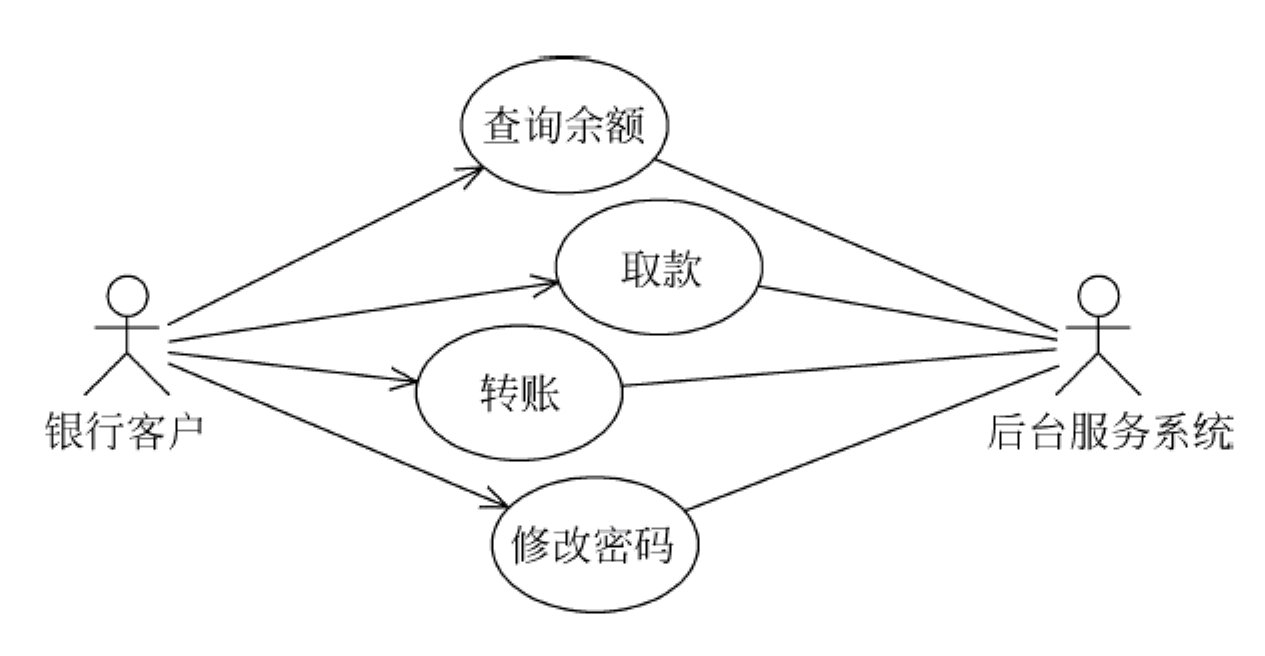


图 12-9 ATM 系统部分用例图

在用例的获取过程中,必须注意:用例之间没有时间顺序,都是参与者通过系统可以获得的服务,它并不表达用例之间的先后关系。用例并不是功能,从功能到用例并不是一对一映射。用例建模另一个常见错误是将一系列相关步骤中的一个步骤定义为用例。例如,输入账户和密码作为一个用例,这仅是登录过程的一个步骤而已。还有,用例必须是由某一个参与者触发而产生的活动,即每个用例至少应该涉及一个参与者。如果存在与参与者不进行交互的用例,就可以考虑将其并入其他用例;或者是检查该用例相对应的参与者是否被遗漏,如果是,则补上该参与者。反之,每个参与者也必须至少涉及一个用例,如果发现有不与任何用例相关联的参与者存在,就应该考虑该参与者是如何与系统发生对话的,或者由参与者确定一个新的用例,或者该参与者是一个多余的模型元素,应该将其删除。

建模过程中应该注意参与者和用例的名称应该符合一定的命名约定,这样整个用例模型才能够符合一定的风格。如参与者的名称一般都是名词,用例名称一般都是动宾词组等。

12.1.5 用例图

用例图展示了用例之间以及同用例参与者之间是怎样相互联系的。它表示参与者使用了系统中的哪些服务(用例),或者说系统所提供的服务(用例)是被哪些参与者所使用的。用例图用于对系统、子系统或类的行为进行可视化,使用户能够理解如何使用系统,并使开发者依据这些用例实现系统。

根据上面分析的参与者与用例,ATM 系统总的用例图如图 12-10 所示。图中"查询余额"、"取款"、"转账"都扩展了"打印凭条"用例。

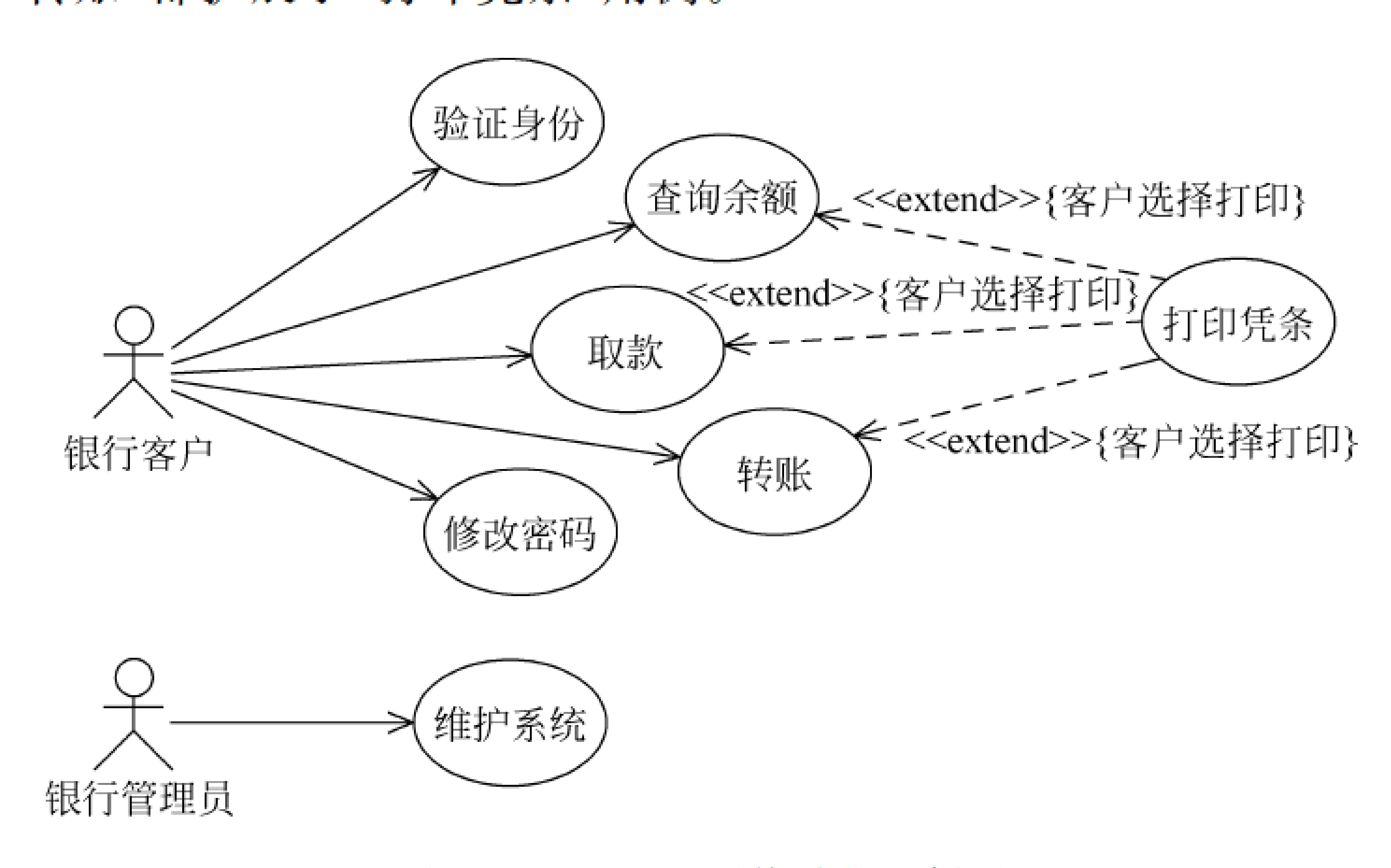


图 12-10 ATM 系统总的用例图

参与者和用例之间的关系用箭头表示,箭头表示在这一关系中哪一方是交互的主动发起者,箭头所指方是交互的被动接受者;如果不想强调交互中的主动与被动关系,可以使用不带箭头的关联实线。在参与者和用例之间的连线并不表示信息流,实际这种信息交互是默认存在的(用例本身描述的就是参与者和系统之间的交互),并且信息流向是双向的,它与关联箭头所指的方向毫无关系。

对于同一个系统,不同的人对于参与者和用例都可能有不同的抽象结果,因而得到不同的用例模型。需要在多个用例模型方案中选择一种"最佳"(或"较佳")的结果,一个好的用例模型应该能够容易被参与系统的各方面人员所理解,并且对于同一用例模型的理解应该是一致的。

12.1.6 用例规约

应该避免这样一种误解——认为由参与者和用例构成的用例图就是用例模型。用例图只是在总体上大致描述了系统所能提供的各种服务,让我们对于系统的功能有一个总体的认识。除此之外,还需要描述每一个用例的详细信息,这些信息包含在用例规约中,用例模型是由用例图和每一个用例的详细描述——用例规约所组成的。用例规约必须要写的内容就是这个用例的实现流程,还有其他一些辅助说明。用例规约基本上是用文本方式来表述的,也有多种组织方式,为了更加清晰地描述事件流程,也可以选择使用状态图、活动图或序列图来辅助说明。只要有助于表达得简洁明了,就可以在用例中任意粘贴用户界面和流程

的图形化显示方式，或是其他图形。也有很多用例规约模板供参考，具体编写可以根据实际需要。下面选择 ATM 例子中较为复杂的转账用例，说明用例规约。

用例编号：UC01

用例名称：转账余额

简要说明：该用例描述银行客户是如何使用 ATM 机来进行转账操作的。

参与者：银行客户，银行后台服务系统

涉众及其关注点：

—银行：准确执行并记录每笔交易，满足客户需求。

—银行客户：得到便捷、快速的服务，准确记录账户信息。

前置条件：客户已通过身份验证。

后置条件：转出金额从客户账户中扣除，并正确追加到转入账户中；正确记录交易日志。

扩展点：打印凭条

触发事件：客户在主菜单中选择“转账”操作后开始该用例。

基本流：

1. 客户在主菜单中选择“转账”操作。
2. 系统提示客户输入转账账号。
3. 客户输入转账账号，该账号必须是国内银行的账号。
4. 系统通过后台服务系统获取转入账户的户主名(没有姓)，显示转入账户名，提示客户输入转账金额。
5. 客户输入转账金额。
6. 系统检查金额在每日限制范围内，且卡内有足够余额，显示转账账号及金额信息请客户确认。
7. 客户确认转账信息无误。
8. 系统启动事务通过后台服务系统执行转账操作，转账成功后记录客户交易日志，并显示转账成功信息。
9. 客户选择打印回执，启动“打印回执”扩展点。

备选流：

*a. 在基本流的任何一步骤执行过程中客户选择“取消(Cancel)”操作：
  系统取消当前操作或撤销当前事务，退出银行卡，用例结束。

*b. 在基本流的任何一步骤执行过程中客户超时(10 秒)：
  系统取消当前操作，返回主菜单，用例结束。

4a. 无效转入账号：
  1. 系统提示转入账号无效，提醒用户认真检查账号。
  2. 客户响应提示，选择重新输入转账账号。

2a. 客户选择取消转账操作：
  1. 系统结束转账操作，用例结束。
  2. 返回到步骤 2。

6a. 转账金额超过限额：
系统提示金额超过转账限额(累计每日不超过 5 万)，返回到步骤 5。
6b. 转账金额超过银行卡账号余额：
1. 系统提示余额不足，要求客户重新输入。
2. 客户重新输入转账金额。
3. 返回到步骤 6。
7a. 客户放弃转账操作：
系统结束转账操作，用例结束。
8a. 转账事务出错(与银行后台服务系统通信失败，或转入账户无法操作)：
1. 系统取消当前操作或撤销当前事务。
2. 系统提示用户相应的错误信息。
3. 用户选择返回，用例结束。

成功场景：
1. 转账成功：基本流。
2. 取消转账：基本流中的某个步骤客户选择了“取消”操作。
3. 操作输入超时：基本流中的某个步骤客户输入超时。

失败场景：
1. 无效转账账号：基本流第 5 步骤中，客户输入无效转账账号。
2. 转账金额超过限额：基本流第 6 步骤中，客户输入转账金额超限。
3. 转账金额超过卡内余额：基本流第 6 步骤中，客户输入转账金额超出卡内余额。

特殊需求：
基本流第 6 步骤的每日转账限额是可插拔式(即可随时替换)的业务规则。

未解决问题：
无。

此用例是示范性的，并且追求彻底详尽，实际是经过多次迭代编写而成。本用例编写得已足够详细，这样做的目的是为了体会详尽描述用例的大量需求细节对后续开发造成的影响，也方便于下面章节的分析。下面针对用例中出现的各个小节进行说明。

1. 基本流

基本流程描述的是该用例最正常的一种场景，在基本流程中系统执行一系列活动步骤来响应参与者提出的服务请求，通常不包括任何条件或分支，所有条件和分支延迟到备选流部分进行说明。建议用以下格式来描述基本流程。

(1) 每一个步骤都需要用数字编号以清楚地标明步骤的先后顺序。

(2) 用一句简短的标题来概括每一步骤的主要内容，这样阅读者可以通过浏览标题来快速地了解用例的主要步骤。在用例建模的早期，也只需要描述到事件流程步骤标题这一层，以免过早地陷入到用例描述的细节中。

(3) 当整个用例模型基本稳定之后，再针对每一步骤详细描述参与者和系统之间所发生的交互。建议采用双向描述法来保证描述的完整性，即每一步骤都需要从正反两个方面来描述：①参与者向系统提交了什么信息；②对此系统有什么样的响应。

在描述参与者和系统之间的信息交换时,需指出来回传递的具体信息。例如,只表述参与者输入了客户信息就不够明确,最好明确地说参与者输入了客户姓名和地址。通常可以利用词汇表让用例的复杂性保持在可控范围内,可以在词汇表中定义客户信息等内容,使用例不至于陷入过多的细节。

2. 备选流

备选流程负责描述用例执行过程中异常的或偶尔发生的一些情况,备选流程和基本流程的组合应该能够覆盖该用例所有可能发生的场景。在描述备选流程时,应该包括以下几个要素。

(1) 起点:该备选流从事件流的哪一步开始。

(2) 条件:在什么条件下会触发该备选流。

(3) 动作:系统在该备选流下会采取哪些动作。

(4) 恢复:该备选流结束之后,该用例应如何继续执行。

备选流的描述格式可以与基本流的格式一致,也需要编号并以标题概述其内容,编号前可以加字母前缀以示与基本流步骤的区别。

3. 用例场景

用例在实际执行时会有很多的不同情况发生,称为用例场景;也可以说场景是用例的实例,在描述用例时要覆盖所有的用例场景,否则就有可能导致需求的遗漏。在用例规约中,场景的描述可以由基本流和备选流的组合来表示。场景既可以帮助我们避免出现需求的遗漏,同时也可以对后续的开发工作起到很大的帮助:开发人员必须实现所有的场景、测试人员可以根据用例场景来设计测试用例。用例场景包括成功场景和失败场景。

4. 特殊需求

特殊需求通常是非功能性需求,它为一个用例所专有,但不适合在用例的事件流文本中进行说明。特殊需求的例子包括法律或法规方面的需求、应用程序标准和所构建系统的质量属性(包括可用性、可靠性、性能或支持性需求等)。此外,其他一些设计约束,如操作系统及环境、兼容性需求等,也可以在此部分中记录。

需要注意的是,这里记录的是专属于该用例的特殊需求;对于一些全局的非功能性需求和设计约束,它们并不是该用例所专有的,应该在补充需求文档中详细记述。

5. 前置和后置条件

前置条件是执行用例之前必须永远为真的条件。在用例中不会检查前置条件,前置条件总是被假定为真。通常前置条件隐含已经成功完成的其他用例场景,例如“登录”。后置条件是用例执行完毕后系统可能处于的一组状态,后置条件应能满足所有涉众的需求。

12.1.7 使用包组织用例

一般小型的系统,其用例模型中包含的参与者和用例不会太多,一个用例图就可以容纳所有的参与者和用例,所有的参与者和用例并存于同一个层次结构中。对于较复杂的大中

型系统，用例模型中的参与者和用例会大大增加，需要一些方法来有效地管理由于规模上升而造成的复杂度。

将用例分组，用包组织管理是最简单的一种手段。根据参与者和用例的特性来对它们进行分类，分别置于不同的用例包管理之下。例如，对于一个大型的企业管理信息系统，根据参与者和用例的内容将它们分别归于人力资源、财务、采购、销售、客户服务这些用例包之下。这样将整个用例模型划分成为两个层次，在第一层次是系统功能总共分为5个部分，在第二层次分别看到每一个用例包里的参与者和用例(图12-11)。

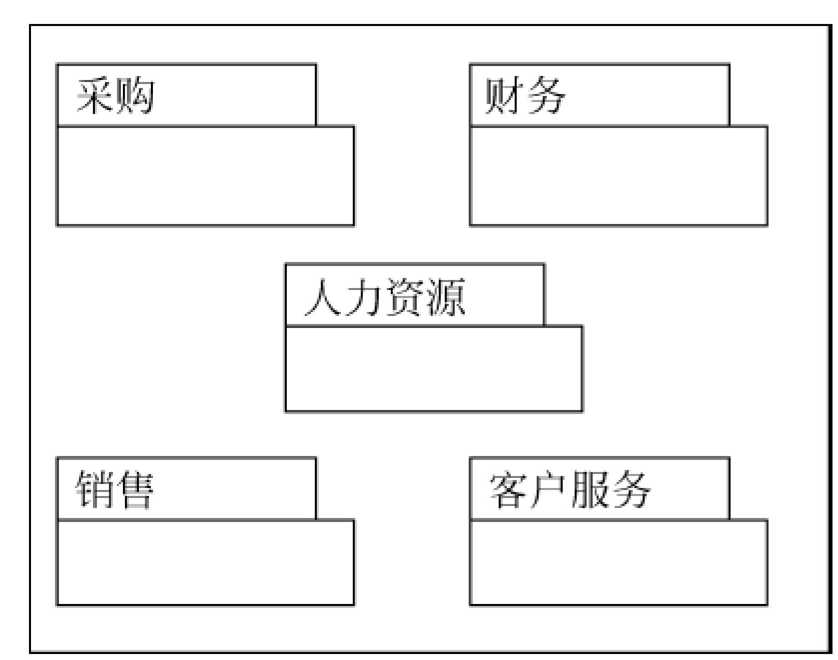


图12-11　用包图组织用例

一个用例模型需要有多少个用例包取决于想怎么样来管理用例模型的复杂度(包括参与者和用例的个数，以及它们之间的相互关系)。UML中的包其实就类似于文件系统中的目录，文件数量少时不需要额外的目录，文件数量一多就需要有多个目录来分类管理，同样一组文件，不同的人会创建不同的目录结构来管理，关键是要保证在目录结构下每一个文件都要易于访问。同样的道理存在于用例建模之中，如何创建用例包以及用例包的个数取决于不同的系统和系统分析员，但要保证整个用例模型易于理解。

12.2　基于类的建模

实际上，用例模型是需求分析成果，并没有使用面向对象思想。用例强调的是系统的活动视图。基于类的建模在用例模型的基础上，开发基于类的领域分析模型，发现构成系统的元素——类和对象、属性、操作及协作等。

领域模型是现实世界中对象的概念透视图，而非软件透视图。其关注的是问题域中事物的可视化表示，而不是诸如Java或C#类的软件对象。分析领域建模的目的是，发现那些表示与问题域相关的事物和概念的类。当建立好用例模型后，由每个用例推导出领域模型，是后继面向对象分析的基础。

12.2.1　识别类

识别类，本质上是一项分析工作，因为寻找出的类可能作为应用程序的构件，这里所说的类也可以简单理解为问题域中参与协作的对象。用例模型中的用例规约提供了发现类的很好依据。从用例规约中所描述的基本流和备选流或一些场景的描述文本，使用“名词法”等方式寻找并抽象出适合的类。通常使用以下策略来发现系统潜在的类。

(1) 外部实体一般都是潜在类，是产生或使用系统处理的信息的来源或消费者。如ATM的转账用例的凭条打印机和银行后台服务系统等。

(2) 组成问题域的事物。例如，系统显示的信息，产生的报告等。

(3) 在系统操作环境内发生的事件。系统范围内的每一项职责都应落实到某个类。如果在用例的执行流程中，存在无对象的职责，就可以发现一些遗漏的类。如ATM执行转账业务，检查当日转账是否超过限额是一项业务规则，应该考虑由哪个对象承担这个职责。

(4) 和系统交互的人员角色。如使用 ATM 执行金融事务的银行客户,管理人员等。

(5) 某个应用相关的组织单元。如部门、组和团队。

(6) 承载问题域的场地。如制造车间或码头。

(7) 系统的组成结构。如传感器、计算机等。

初始时常从用例规约中列举出所有的名词,作为候选类。一些类可能仍未被发现,一些类可能在后继分析中被淘汰。这些都没有关系,分析都是多次迭代,不断完善的过程。有了这些潜在类的列表后,去除重复的或表达意义相同的名词,删除与系统无关的、处于系统边界外的名词,把剩下的名词进行归纳或抽象,得到需要建模的对象。

Coad 和 Yourdon① 提了 6 项选择这些候选类的特征,在考虑分析模型时,筛选潜在类时应参照这些特征。

(1) 保留信息。只有当类中保存相关信息才能保证系统正常工作时,潜在类在分析过程中才是有用的。

(2) 所需服务。潜在类必须具有一组可确认的、能用某种方式改变类的属性值的操作。

(3) 多个属性。在需求分析过程中,焦点应在于"主"信息;事实上,只有一个属性的类可能在设计中有用,但是在分析活动阶段,把它作为另一个类的某个属性可能更好。

(4) 公共属性。可以为潜在类定义一组属性,这些属性适用于类的所有实例。

(5) 公共操作。可以为潜在类定义一组操作,这些操作适用于类的所有实例。

(6) 必要需求。在问题空间中出现的外部实体,或者任何系统解决方案的运行所必需的信息,几乎都被定义为分析模型中的类。

考虑上面所选择的名词列表,只有全部或几乎全部满足这些特征的名词才能作为进一步分析的潜在类。判定潜在类是否包含在分析模型中似乎有点主观,而且后面的评估可能会舍弃或恢复某个类。然而这是分析建模的第一步聚,必须进行决策确定出潜在类。

以 ATM 的转账用例为例,首先从 ATM 的转账用例规约中,列举出所有名词:银行客户、系统(ATM)、"转账"操作、转账账号、后台服务系统、账户、户主名、转账金额、每日限制范围、余额、事务、日志、提示信息、回执。

然后整理名词列表,针对相同意义的名词或短语,选择表达更好的。根据上述 6 个特征筛选潜在类,如表 12-1 所示。

表 12-1 潜在类筛选

潜在类	所适用的特征
银行客户	拒绝:6 适用但是 1,2 不符合
账户	接受:所有都适用
户主名	拒绝:3 不符合
系统(ATM)	接受:所有都适用
"转账"操作	接受:所有都适用
每日限制范围	接受:所有都适用
余额	拒绝:3 不符合,是账户的属性
事务	接受:所有都适用
日志	接受:所有都适用
凭条打印机	接受:所有都适用

① Coad P., Yourdon E. Object-Oriented Analysis. 2nd ed.. Prentice-Hall, 1991

银行客户是外部的参与者，是代表现实世界的参与者，在银行系统中以账户形式存在，因此只需要对账户对象建模为类"Account"。检查与账户相关的名词：转账账号、户主名、余额。转账账户只是账户在转账过程中承当的一种角色，本质上还是账户。而户主名、余额都是账户的信息，不应该建模为类。

系统(ATM)代表一个自动取款机整体，但从概念上创建一个类表示机器是很有用的，建模为"ATM"。

"转账"操作是客户使用 ATM 执行的一项金融事务，类似的事务还有查询、取款和修改密码等。这些事件有其职责，如转账事务需要知道转入账户、转账账户和转账金额。因此，转账操作建模为转账事务"TransferAccount"。

另一个名词"事务"，是这些金融交易的统称。这是由于这些金融交易操作都必须以事务方式与后台服务系统交互，以免中途出现故障导致数据不一致。查询、取款、转账和修改密码都是金融事务的一种，可以统一抽象为一个事务类"Transaction"。虽然在场景模拟过程中，不会使用到一个抽象的类，但在后续类的建模时会使用到。

后台服务系统也是 ATM 的一个外部参与者，完成金融交易需要银行后台服务系统的协助。在系统中为它建模为"BankServer"，让它作为后台系统在 ATM 机中的接口代表，也能在系统中很好地模拟 ATM 与后台服务系统的交互。

每日限制范围是一项业务规则，是能影响业务执行的对象，而且有可能在不久的将来会有所改变，是个有效的候选类。ATM 的业务规则还有许多，可以统一简化为一个业务规则类。

很明显，日志和回执都是业务领域的对象。

至于提示信息则需要根据实际情况，如果在交互过程中只是简单的字符串信息，则无须看成一个类，如果准备传递很复杂的组合信息，则需要作为一个类。

12.2.2　CRC 建模

类职责协作(Class-Responsibility-Collaborator，CRC)卡建模[①]是一种简单且有效的面向对象的分析技术。在一个系统开发项目中，包括用户、系统分析员和开发者，在建模和设计过程中，经常应用 CRC 卡建模，使整个开发团队普遍的理解形成一致。在 CRC 卡建模过程中使用真实存在的卡片，而不是计算机或一张薄纸。它用厚纸板或普通的索引卡做成，每个卡片代表一个类。一张 CRC 卡由类(Class)、职责(Responsibility)和协作(Collaborator)3 部分组成(图 12-12)。

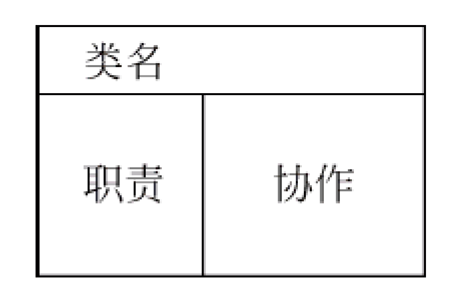


图 12-12　CRC 卡

类代表许多类似的对象，而对象是系统模型化中关注的事物。对象可以是人、地方、事情或任何对系统有影响的概念。如大学教务管理系统中，学生、公选课或班级都是类。类名一般列在 CRC 卡的顶部。

职责是类需要知道或需要做的任何事物。这些职责是类自身所具备的知识，或类在执行时所需要的知识。如学生有学号、姓名和联系方式，这是学生自身具备的知识，学生选择公选

① 余金山. 实时 UML 与 Rational Rose RealTime 建模案例剖析

课,申请缓考和查询学期所选课程成绩,这些是学生需要做的事情。职责写在 CRC 卡的左边。

一个类要实现某一职责时,却发现没有足够信息去完成该职责,这时就需要其他类的协作。协作是指为获取消息或协助执行活动的其他类。协作就是在特定情形下,与指定的类按照设想共同完成同一个(或许多)场景。如学生要完成“选择公选课”这一职责,需要知道这门公选课中是否有空名额,因此需要与公选课类协作,以完成“选择公选课”职责。协作的类名在 CRC 卡的右边——与完成的职责相对应的位置。

进行 CRC 卡建模的步骤大致如下。

(1) 分析类的职责。

(2) 寻找协作者。

(3) 组织建模团队场景模拟。

(4) 迭代建模,最终排列卡片模拟系统运行。

CRC 卡建模时,首先为领域中所有的类建立初始卡片,然后组织建模团队,包括分析员、用户和领域专家,进行用例中的场景模拟。对用例场景多次迭代分析,进行 CRC 建模,发现新类、修改或补充类的职责及协作,最终让这些卡片无误地表达出系统的运作方式。

场景模拟最好是以小组的形式来做。小组中每个成员赋予一个类(或少量的几个类),然后每个人大声说出这个类当前正在做的事情,以此来说明自己扮演的角色。在场景模拟过程中,要在 CRC 卡上记录模拟中与这个类有关的所有事情,即责任应该是什么,要与其他哪些类合作。

使用 CRC 卡建模时,将识别的潜在类建立 CRC 卡片。如图 12-13 显示了转账用例中的潜在类。

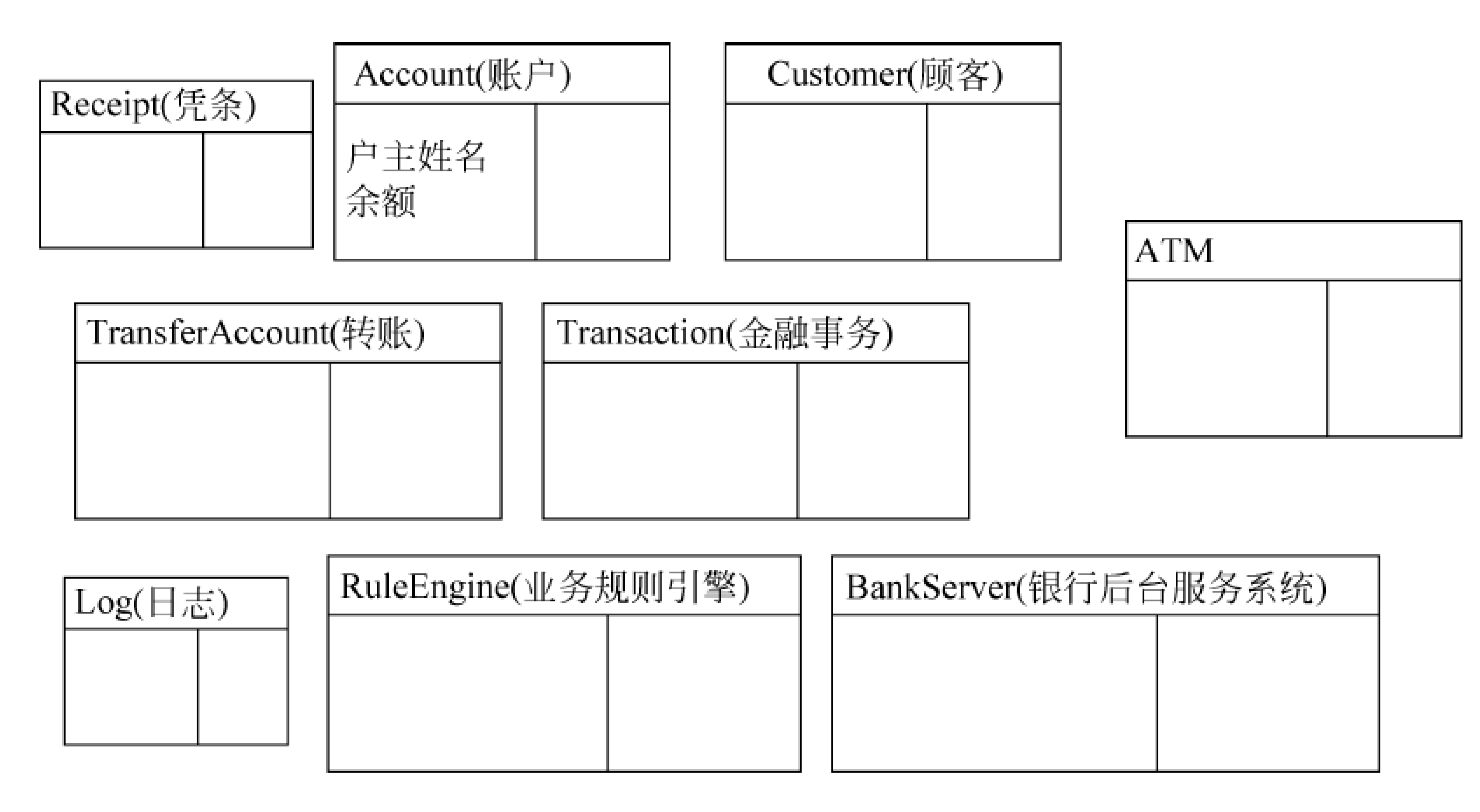


图 12-13 转账用例中的潜在类

对于其他的用例同样根据规则及业务实际进行筛选,建立类的 CRC 卡。

读卡器、出钞器、银行卡、键盘、显示屏和凭条打印机,都是 ATM 的外部设备,分别对这些硬件设备建模,作为系统中与外部设备交互的接口(图 12-14)。

另外还有一些金融事务对象,如取款事务,用于处理用户取款事务的业务逻辑;查询余额事务,用于执行账户查询的业务逻辑;修改密码事务,用于执行账户密码修改的业务逻辑(图 12-15)。

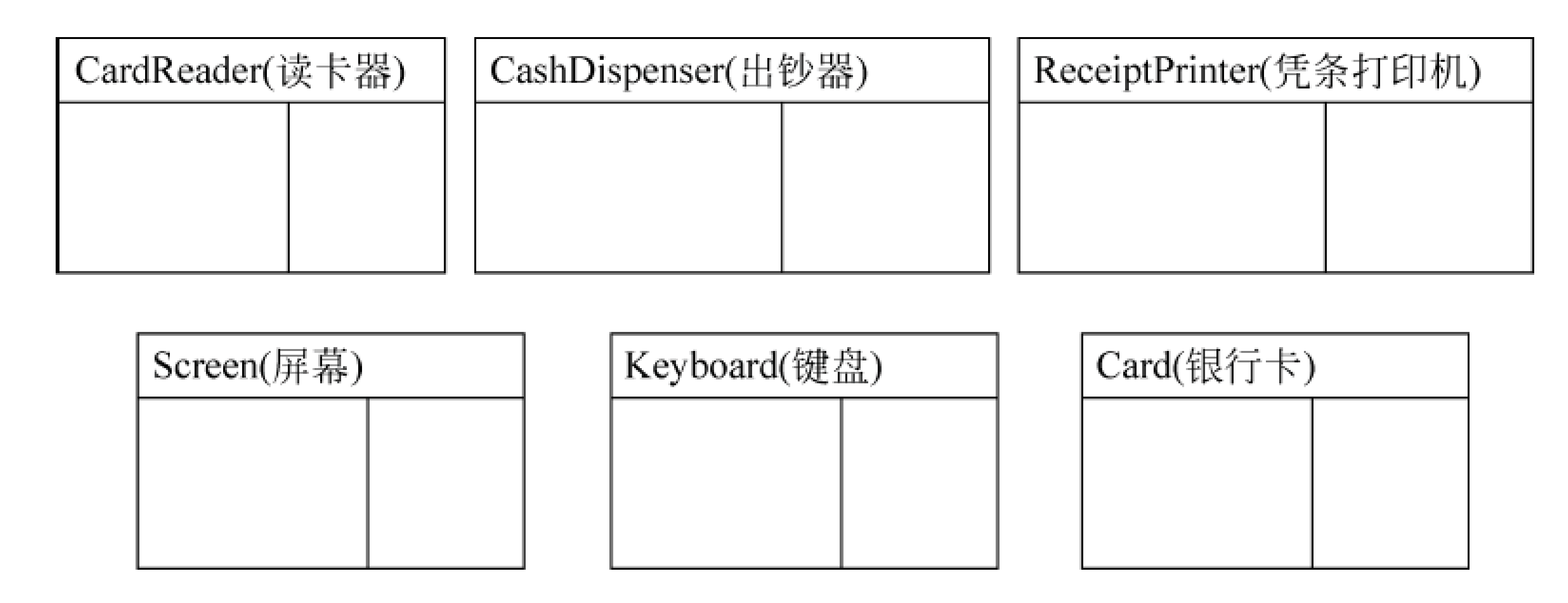


图 12-14　问题域中的自然对象建模为类

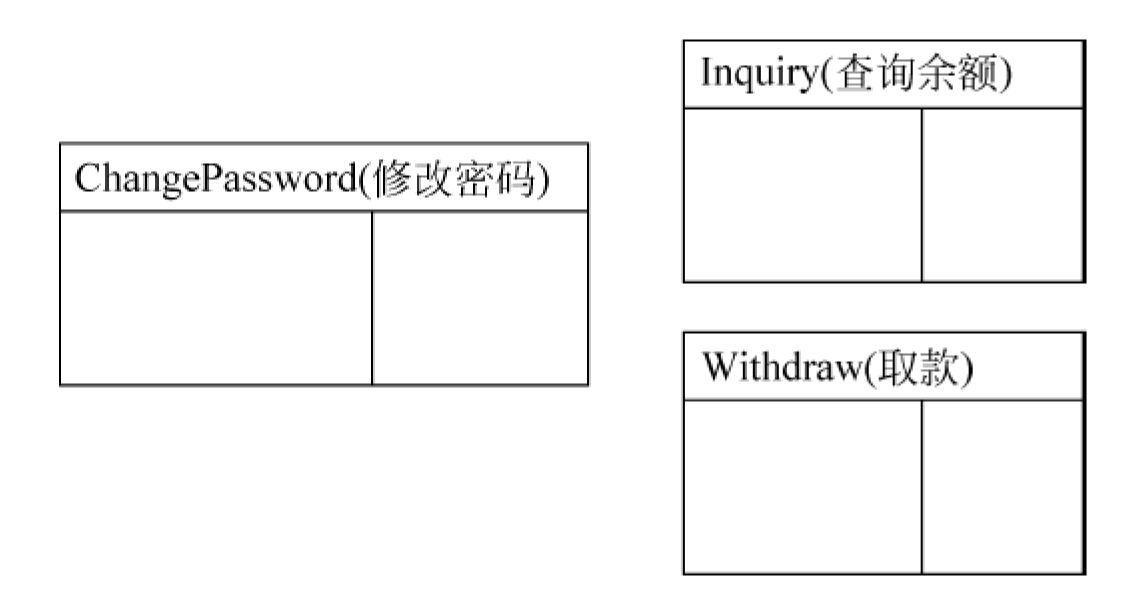


图 12-15　业务逻辑对象

12.2.3　分析类的职责

职责包括以下两个方面内容。

- 属于对象，由对象自行维护的信息。
- 对象能够执行的行为，归属于对象的部分系统责任。

对象本身就是以类的形式表示数据和功能的组合，类所维护的信息是它的数据，要执行的行为是它的功能。类的职责体现了一个对象在系统中的作用和地位。分析类的职责可以从以下几个方面着手。

(1) 分析列举出来的类，应该存储些什么信息。类的名称暗示着它的职责。从类的简要说明中寻找类天然具有的信息。

(2) 列出需求中所有的动词。用例规约是对需求的详细描述，再次阅读列出所有的动词，显示这些动作是系统内某些对象必须完成的行为。

(3) 分析用例实现的过程，设想类的整个过程中承担的责任。从用例中分析出来的类，就是为了帮助实现用例，如果在用例的实现过程中不承担职责，那么类也就没有存在的意义。

再次阅读转账用例规约，列出所有动词，分析这些动词中哪些明显代表了系统内某些对象必须完成的动作。同时注意用例规约中提到的信息，发现在系统内某些对象必须维护和处理的信息。

图 12-16 是转账用例中的类的部分职责示例。

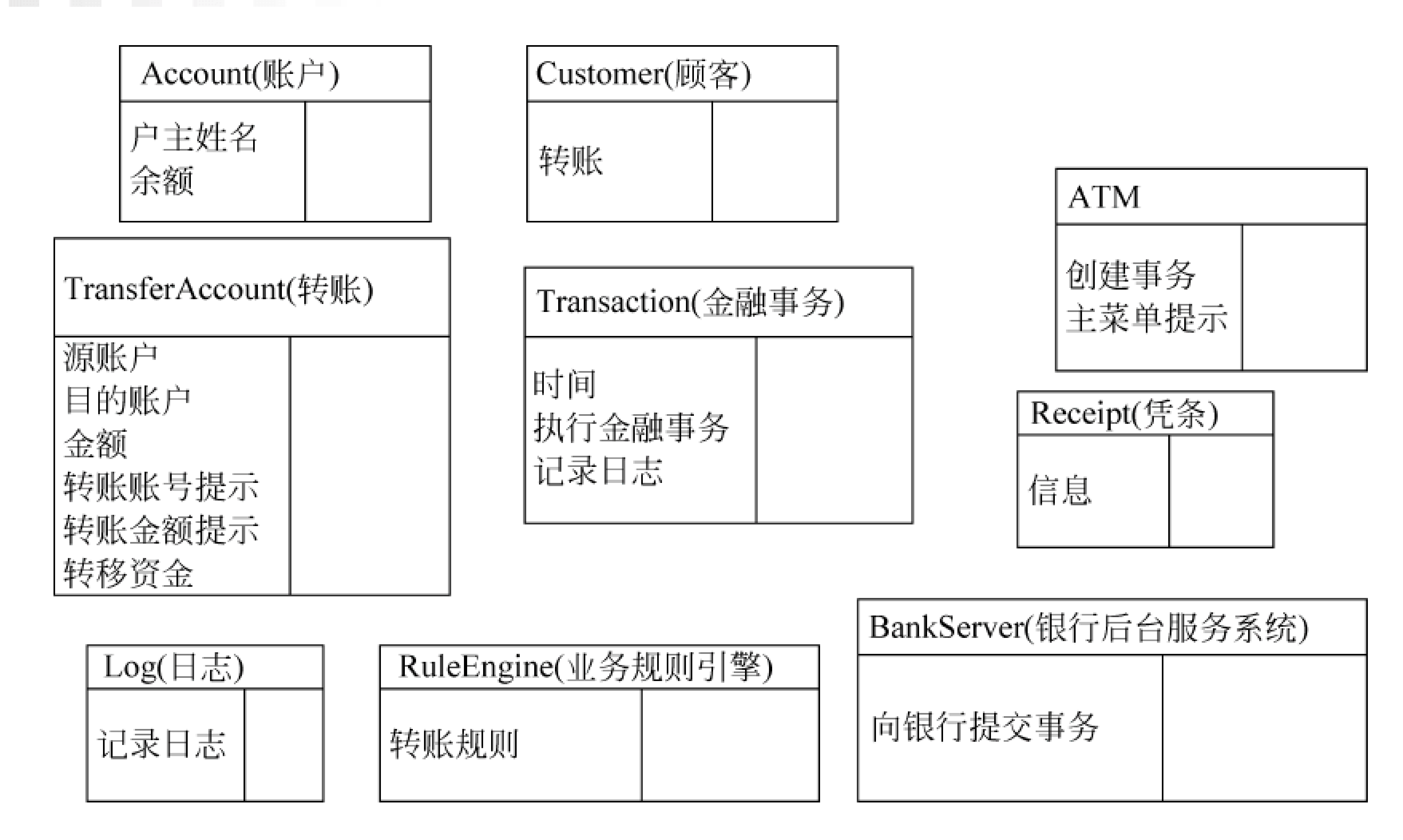


图 12-16 转账用例中的类的部分职责

12.2.4 寻找协作者

对象自己能够履行具体的职责，也需要其他对象的帮助。仅当类 A 为类 B 完成某些事情时，类 A 才显示为类 B 的协作者。定义职责和寻找协作者是个高度迭代的过程，没有一定的顺序。类的许多职责往往是通过协作发现的。当一个类需要和其他类协作时，意味着第二个类现在有了完成该协作的职责。因此，当发现职责时，需要定义协作，同时，当定义协作时，常常会发现新的职责。

为了确定协作，可以对每个类的每个职责询问下列问题①。

(1) 类本身能够履行这个职责吗？

(2) 如果不能，那么它需要什么？

(3) 从其他什么类中它能够获得所需要的东西？

弄清楚这些问题的答案，也就找到了职责的协作者。我们可以顺着用例的实现顺序寻找类的协作者。图 12-17 是 ATM 类和 Transaction 类的协作示例。

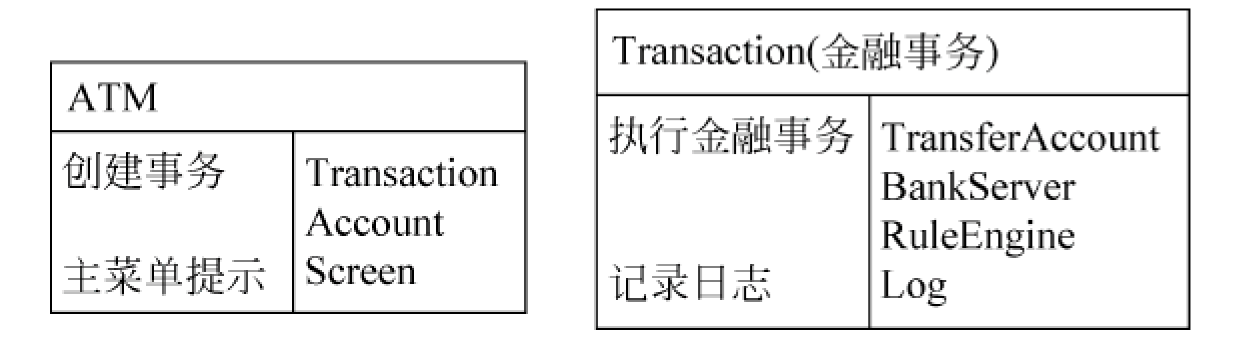


图 12-17 ATM 类和 Transaction 类的协作

12.2.5 通过场景验证 CRC 模型

当一个完整的 CRC 模型开发出来后，建模团队可以使用 CRC 卡模拟系统的某个用例场景，以便验证开发出来的 CRC 卡的正确性和合理性。

---

① Rebecca Wrifs-Brock，et al.. 面向对象软件设计经典. 张金明等译. 2003

首先建模小组成员各自拿到一部分 CRC 模型卡片，扮演系统中的对象角色。卡片按照协作分开，即不得有两张存在协作关系的卡片在同一个人手里。然后选择某个用例场景，由组长细致阅读用例场景过程。查看启动场景的类，将令牌交给拥有这个类卡片的人员，类卡的拥有者说明卡上记录的启动事件的职责，并顺着场景的执行过程，根据卡上记录该启动事件的协作者，寻找下一个职责承担的类，将令牌传递给拥有该类卡片的人员。依次传递，直到场景结束。当令牌传递时，类卡的拥有者需要说明卡上所记录的职责，让小组共同确定职责是否满足用例需求。当卡上所记录的职责或协作不能满足用例时，就需要修改卡片。如果发现有些职责找不到合适的类负责，则需要建立新的卡片。如图 12-18 所示，每次演练一个场景，并且一边演练一边更新类模型中的对象或增加其职责，使其满足场景的需要。

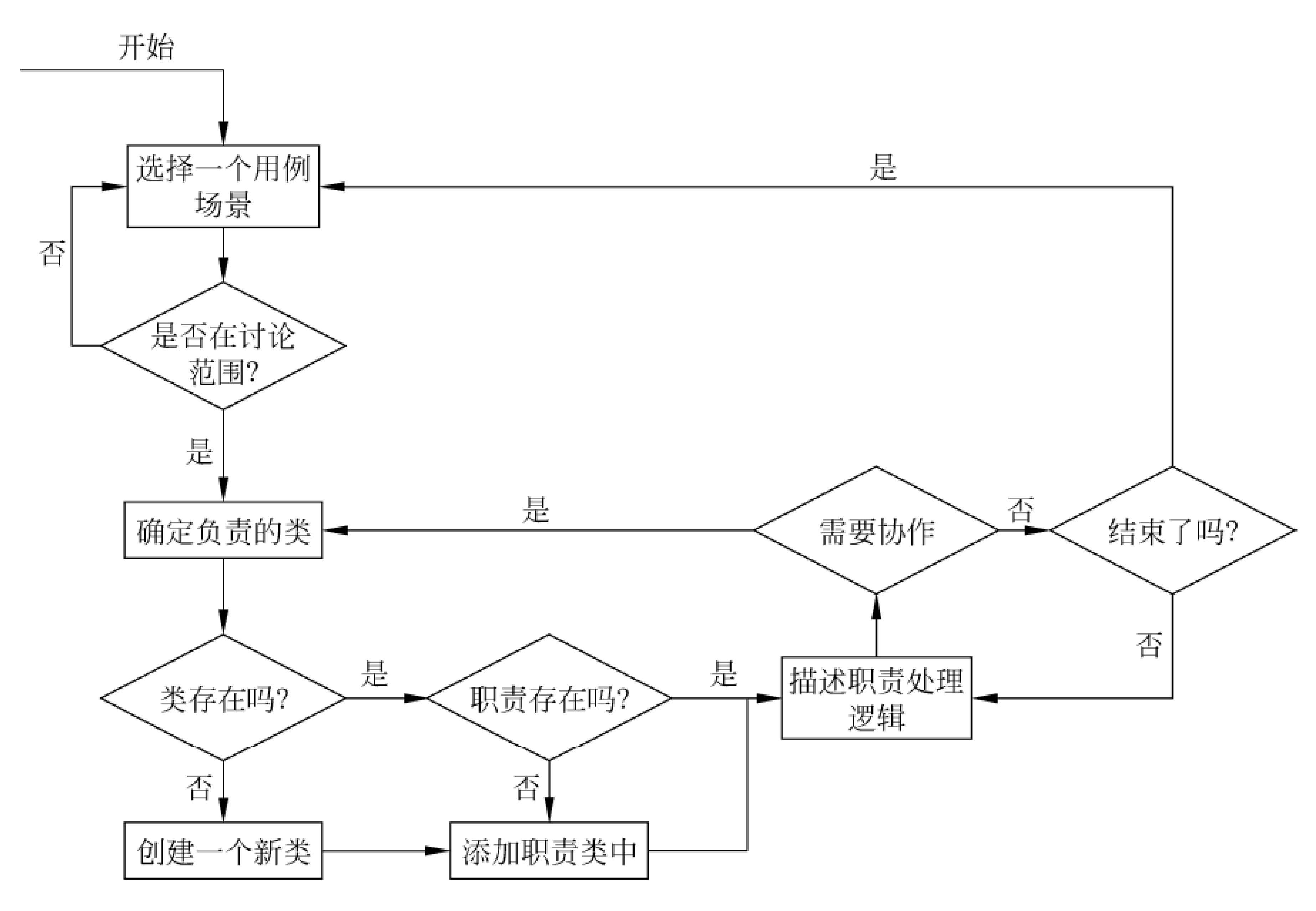


图 12-18　用例场景演练

如 ATM 转账用例，组长首先看到客户选择转账操作启动用例，因此把令牌传给拥有 Customer 卡片的人员，暗示着该用例由 Customer 对象启动。卡片的拥有者检查卡上记录的转账职责，并说明。追踪到协作者是 ATM，则将令牌传递给拥有 ATM 卡片的人员。ATM 让 TransferAccount 事务对象实现转账操作。TransferAccount 对象请求 Screen 显示提示用户输入转账账号的信息，并通过 Keyboard 对象获取用户输入的账号，同时请求 BankServer 验证输入账户的合法性，通过 BankServer 对象获取转账账户名。同样，利用 Screen 和 Keyboard 获取用户输入的转账金额。根据 RuleEngine 对象提供每日转账限额规则检查客户转账是否超限，最后将转账信息提交给 BankServer 对象执行转账。转账成功后记录日志 Log 并通知客户，场景结束。

在场景模拟过程中，应该质疑和记录场景执行过程中的每一个细节，以免到开发时才临时做决定，造成不良影响。

12.2.6 CRC 建模的优缺点

CRC 建模可以很有效地提高分析质量。一方面，在整个建模过程中，有最终用户或领域专家的参与，而不是分析人员自己的猜测和推导。分析和定义需求的人应该是那些理解业务的人。而系统分析员本身对于问题域的知识有限，仅靠与用户沟通了解业务存在片面性。除非参与到业务中完成用户要做的工作，否则不会比真正完成这些工作的人更了解业务。另一方面，分析人员与用户可能都不具备很好的沟通能力，分析人员是拥有技术方面的能力，而用户只是自己业务领域的专家。有时可能用户无法清楚地表达需求，造成分析人员误解，也可能是分析人员误导了用户。面向对象分析方法本身采用的就是普通人的认识世界的方法，尽量让用户参与系统分析对建模有如下好处。

(1) 增加了系统开发中用户参与的可能。促进了用户和开发人员之间的交流。

(2) CRC 卡建模简单而且廉价。CRC 卡很简洁，只需要一些卡片来演化整个系统的执行。

(3) CRC 建模让用户对系统有很好的全局观点。

(4) CRC 建模直接引导我们进入类建模。类建模是面向对象分析与设计的核心。

(5) CRC 建模促进形成公用项目词汇表，确定业务领域中的关键术语，统一对业务领域的认识。

但是 CRC 建模同样也存在一些缺点。

(1) CRC 建模让系统分析人员不习惯。有些系统分析人员更喜欢一个人安静地思考问题，而不是一群人的讨论。

(2) 让一些用户或是领域专家聚到一起开会是一件困难的事。用户本身有自己的工作，参与分析增加了他的职责。

(3) CRC 卡也存在缺陷。最重要的是无法在卡上详细记录业务逻辑，对于继承、联合和关联概念，CRC 卡也无法表示。

(4) CRC 卡不是万能的，只是面向对象开发过程的一个步骤，需要把已经完成的 CRC 模型转变为类模型，然后用类模型生成系统分析文档。

12.3 类模型

类模型是面向对象分析与设计的支柱。类模型展示系统的类以及它们的相互关系(包括继承、聚合与关联)以及类的方法和属性。类模型是在 CRC 模型基础上扩展而来的，给出了更多细节、范围更宽，是后期编码的主要依据。

开始进行类模型建模的最简单方法就是直接把 CRC 模型转换成 UML 类图。CRC 模型给出了系统最初的类、它们的职责以及这些类之间的相互关系(以协作者列表形式展现)。CRC 模型提供了对系统的一种描述，但它没有提供实际构建过程中需要用到的细节。

对于 CRC 模型中的每张卡片，都在类图中创建一个具体类，当然参与者的卡片除外。另外，CRC 卡上的协作者隐含着类间关联、聚合或依赖等的关系。两个具体类间有协作，就应该建模为关联。卡中的职责通常建模成属性或方法。

另外还可以使用交互模型对用例场景的逻辑进行梳理和验证，可以发现用例实现过程缺少的类，或补充类的属性或职责，修正类与类之间的关系。

12.3.1　类、属性和方法

类是对象的模板，是组成面向对象应用程序的主要构件。类用矩形框建模，最上部是类名，中部是类的属性，底部是类的方法。系统初始的类可以由 CRC 模型中的领域对象直接转换。类所知道的事情，即为类的属性；需要执行的职责，即是类的方法。例如 ATM 系统里的账户有账号、余额、密码和用户姓名等属性，账户还能执行查询账户余额、取款和修改账户密码等事情，这些是类的方法（图 12-19）。

BankAccount

− 账号
− 户名
− 余额
− 密码

+ 查询()
+ 取款()
+ 修改密码()

图 12-19　银行账户类

12.3.2　类的层次结构

不同类之间存在相似性。常常两个或多个类共享同一个属性或同一个方法。继承是类的层次结构。继承建模表示“is a”和“is like”关系。

在 ATM 系统中，转账、查询、取款和修改密码等金融业务，需要通过银行的后台服务系统完成，必须以事务的形式向后台提交请求。因此，需要创建一个事务类（Transaction），在事务类中实现开始事务和结束事务方法，转账、查询、取款和修改密码类都继承于事务类（图 12-20）。如果后面分析还有一些公共方法或属性，都可以移到事务类中。

原来金融事务类与 ATM 类还有其他设备接口类的相同语义的协作关系，也可以统一到事务类中，简化类的模型。

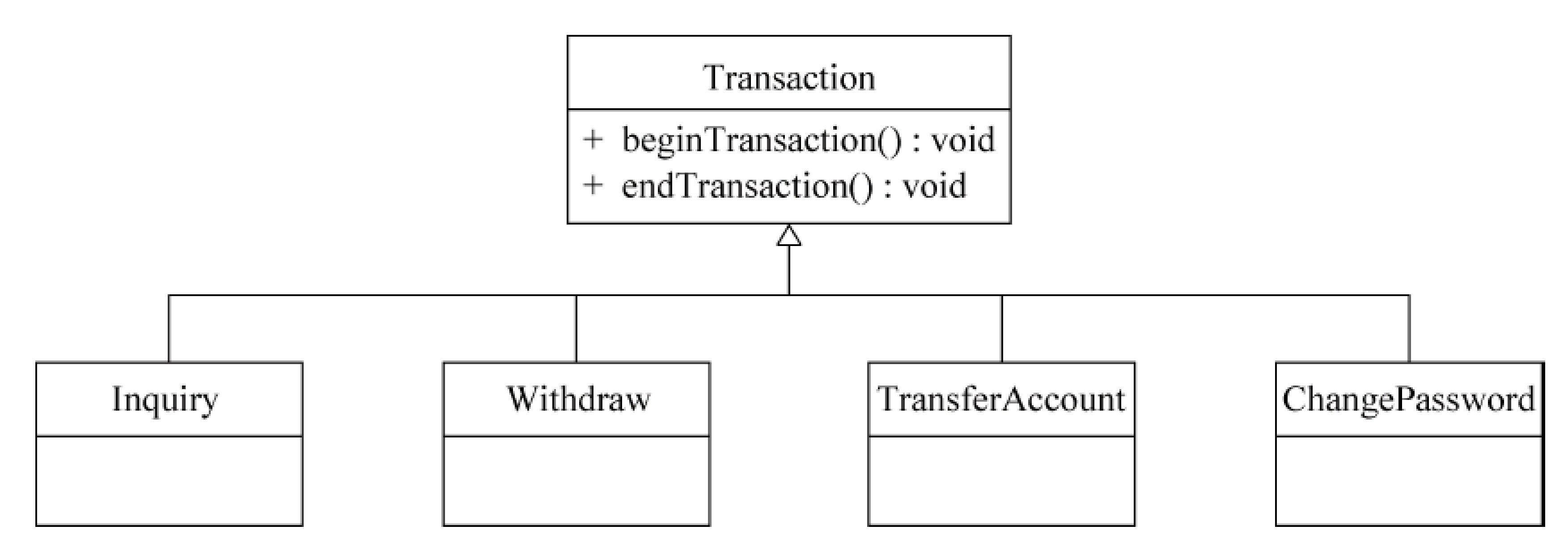


图 12-20　事务类的层次结构

12.3.3　关联

在系统中，对象相互间存在关系，称为关联，这是类间存在的最普通的关系。两个类间存在关系，也就是意味类间存在耦合。需要分析出类间所有关系，但要尽量减少类间关联，以达到建立高内聚低耦合系统的目的。

在 CRC 卡模型中，如果两个类存在协作，则说明两个类间存在关联。关联表示为连接两个类的一条细线。关联建模包括关联方向性、末端角色名、多重性和关联名。关联名主要是用简单的词说明两个类的关系，如日志类记录每个事务。

仅用一条连线表示两个类之间存在联系是不够的，如开发团队与成员间存在联系，只有一条连接似乎表示一个开发团队只有一个成员的关系。因此，在分析关联时，还要关注类间的多重性（multiplicity），也就是某个类有多少个实例对象与另一个类的单个对象关联，表示

多重性的方法是在参与关联的类附近的关联线上注明多重性。如图 12-21 所示，一台 ATM 可能启动 0 个或多个事务活动，而同一时刻一个事务活动只属于一台 ATM 机器。

当一个类和另一个类发生关联时，类通常在关联中扮演某种角色。可以在图中靠近类的地方的关联线上标明该类的在关联中的角色。在 ATM 与银行账户的关联中，银行账户所扮演的角色就是当前在 ATM 开展业务活动账户的角色，因此在关联线靠近 BankAccount 的一端，标明当前账户(CurrentAccount)。关联的表示方法如图 12-21 所示。

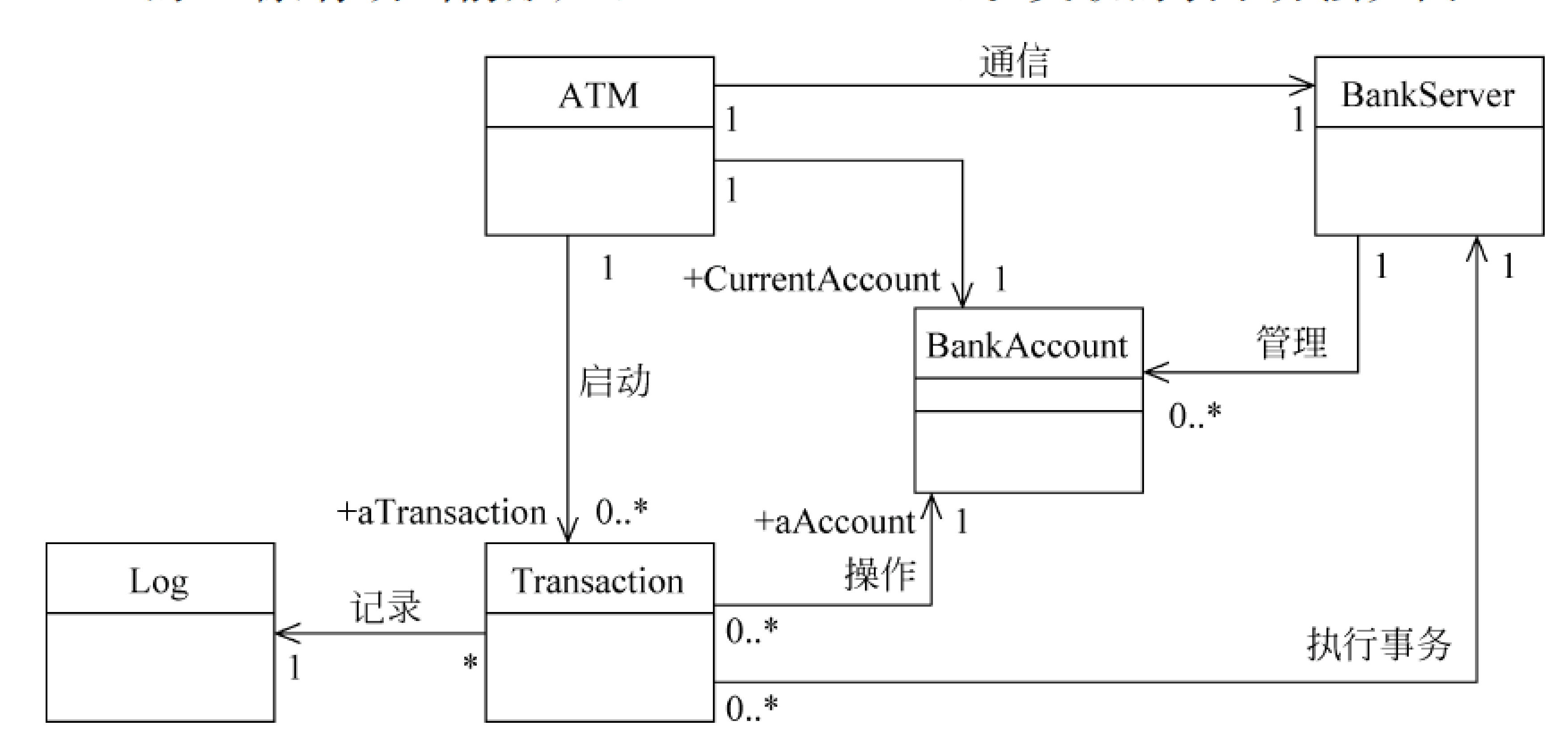


图 12-21 ATM 的部分类的关联

12.3.4 聚合

有时某个对象是由其他对象组成的，如汽车是由发动机、轮胎、变速箱等组成的；一个团队由两个以上雇员组成；一份订单包括一个或多个商品。这些都是聚合的例子。聚合表示“is part of ”的关系。发动机是汽车的一部分，雇员是团队的一部分，商品是订单的一部分。

组合是一种更强形式的聚合。在组合中，“整体”负责各组成部分，每个“部分”仅和一个“整体”对象相联系。在 CRC 模型中，建模了读卡器(CardReader)、出钞器(CashDispenser)、屏幕(Screen)和凭条打印机(ReceiptPrinter)外部设备接口类，它们是组成 ATM 的设备类。这些外部设备都是组成 ATM 的一部分，因此它们之间为组合关系(图 12-22)。

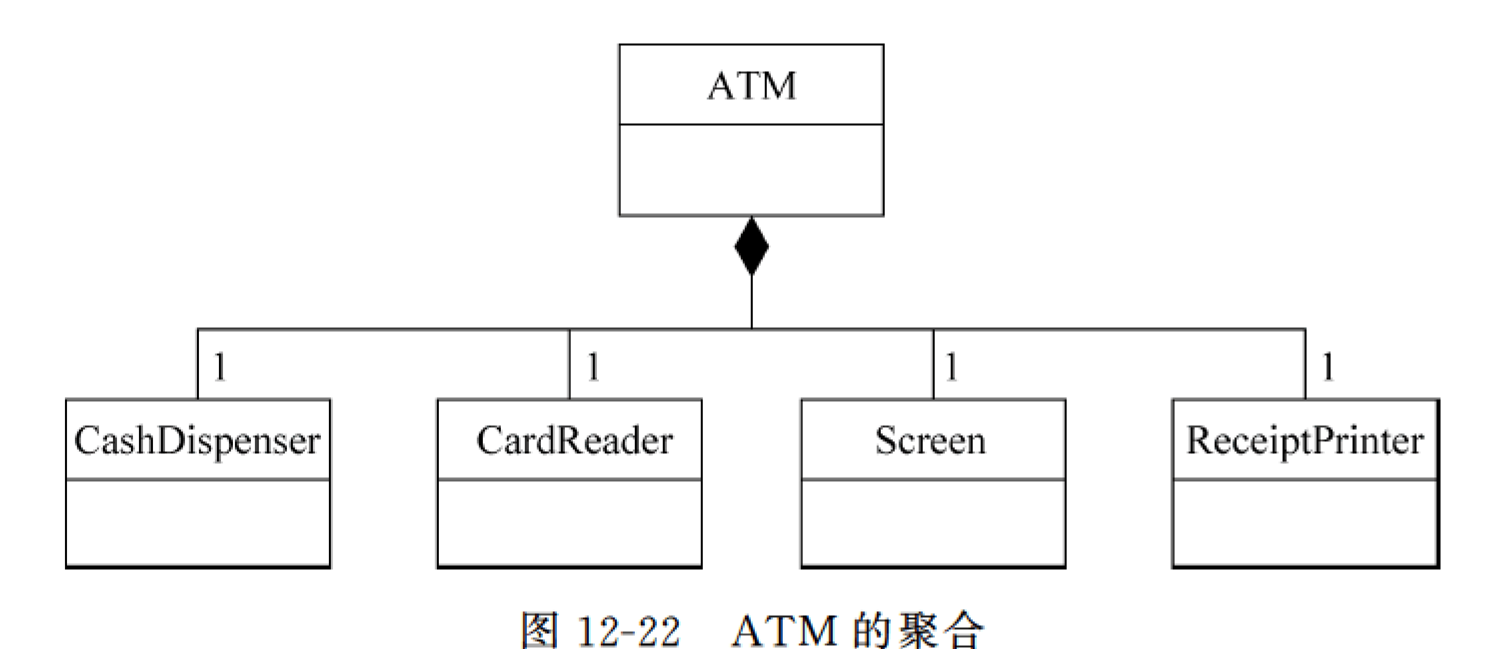


图 12-22 ATM 的聚合

12.3.5 类图

将 CRC 模型包含的信息转变成最初的类模型后，经过交互图建模的动态分析，加上类图的静态细节描述，使类图包含更丰富的信息。在实践中，这种互补的静态视图和动态视图建模是并行创建，交替开展的。

经过对各个用例分析，对初始类图归纳优化，ATM 系统的总类图如图 12-23 所示。ATM 包含读卡器（CardReader）、钞票分配器（CashDispenser）、屏幕（Screen）和凭条打印机（ReceiptPrinter）部件。ATM 必须也与银行后台服务系统（BankServer）通信完成顾客金融事务。顾客的账户信息（BankAccount）归银行后台服务系统管理。当顾客插入银行卡时，ATM 的当前处理账户就是该名顾客的银行账户，一旦顾客有任何金融事务（Transaction）的请求，ATM 启动相应的金融事务对象完成顾客的请求。顾客进行的金融事务主要有转账（TransferAccount）、取款（Withdraw）、查询金额（Inquiry）和修改密码（ChangePassword）。每次操作的金融事务都是记录到 ATM 的日志（Log）中备查。同时，某些金融事务操作时也要满足业务规则（RuleSet）的限制。修改密码严格来说不能算是金融事务，但与金融事务统一起来，会为后期的设计与编码带来方便。

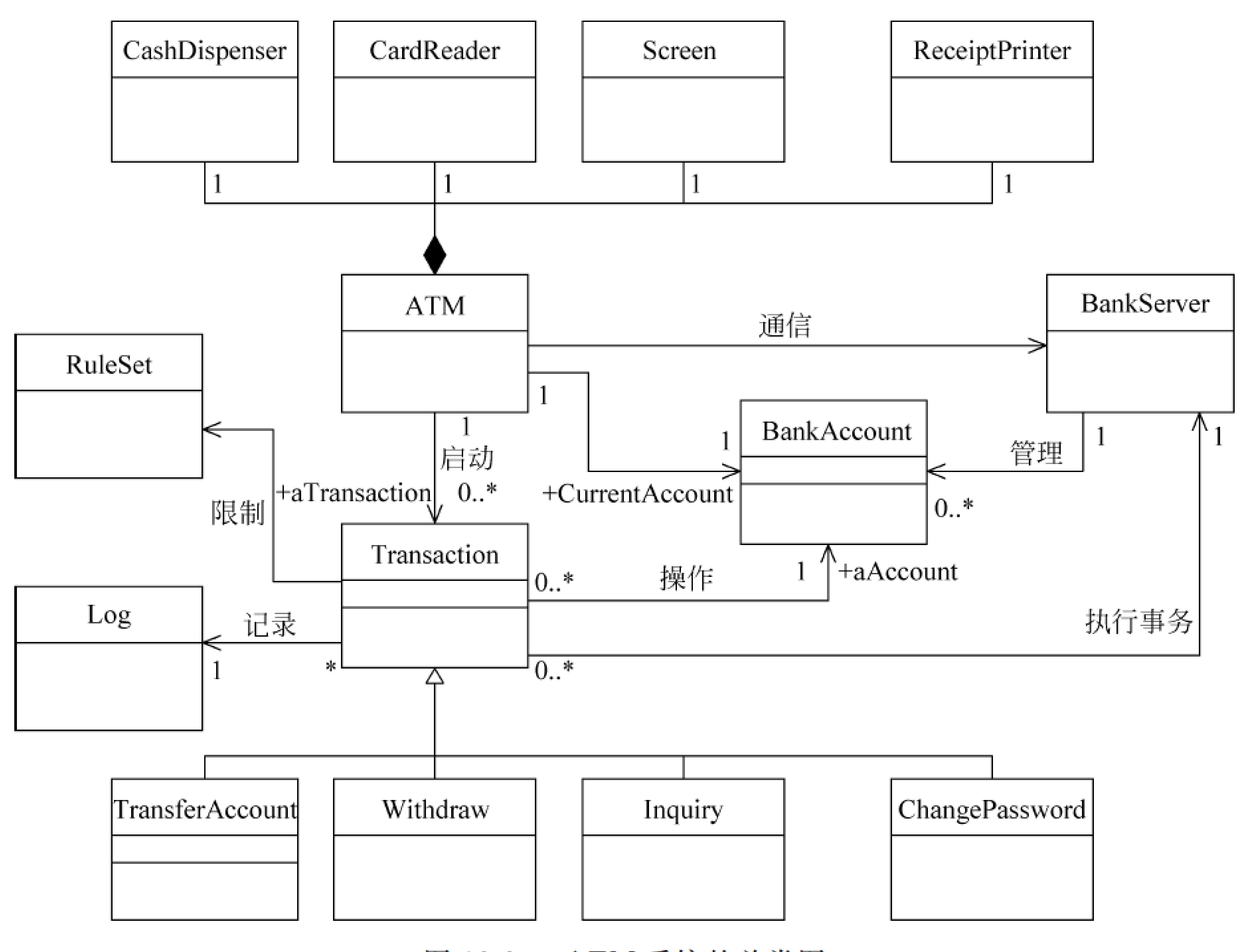


图 12-23　ATM 系统的总类图

12.4　动态交互建模

CRC 模型确立了完成业务需求的潜在参与对象，以及对象相互的静态协作关系。但这些对象如何协作实现具体的业务任务并没有描述。顺序图、协作图和活动图对参与者使用系统的场景进行逻辑建模。使用场景可能是用例基本流的一部分，也可能是备选流。它们以可视化的方式对系统的某一场景的逻辑流程进行动态建模，使分析员能记录并验证逻辑执行过程。

每种图都有其优点，而分析员也各有偏好，因此没有绝对的“正确”的选择。然而场景交互强调的是顺序，因此顺序图有更强的表示能力。采用顺序图可以更方便地表示对象协作发送消息的顺序，仅需要由上至下阅读即可。而对于协作图，则必须查阅消息的顺序编号才能理解协作的顺序。但由于顺序图采用栅栏格式描述交互，在右侧添加新创建的对象，底部

添加消息,容易占用纸上空间。协作图则空间使用效率较高,允许在任何位置放置对象,消息直接标注的连接线上。

12.4.1 顺序图

顺序图在分析或是设计过程中都可能使用。每个用例至少要做一个顺序图,有时也为一个用例创建多个顺序图。顺序图应和用例规约的叙述流程一致,如果开始用例绘制顺序图时有问题,那么很可能是编写的用例规约不正确,应该重新考虑用例规约的逻辑。

因此,使用顺序图分析用例动态实现过程的优点如下。

(1) 是验证用例逻辑及使逻辑清晰的好方法。

(2) 根据用例规约描述,推测用例的可能实现过程。

(3) 发现系统瓶颈的一种很好的机制。

(4) 检验分析模型,确认是否缺少一些类。如果还有职责无法落实到某个对象,则需要补充一些对象,或在一些类中补充一些方法或属性。

(5) 帮助发现处于系统边界的主动对象。如某个对象的服务来自系统边界的信号引发执行,则为主动对象。如有些系统定时进行自检,则自检对象为主动对象。也可以将外部异步信号设计为一个参与者,由这个参与者触发自检用例。

(6) 对于设计、编程、测试人员来说,顺序图是一个参考文档,提供了某一项功能可追踪的执行路线。

在使用绘制顺序图时首先要说明顺序图所绘制的用例场景,如有必要,可以把描述场景的文本列在顺序图旁。

前面已经有了转账用例的用例规约,以及 ATM 的 CRC 模型。在 CRC 模型中参与转账用例协作的类如图 12-24 所示。

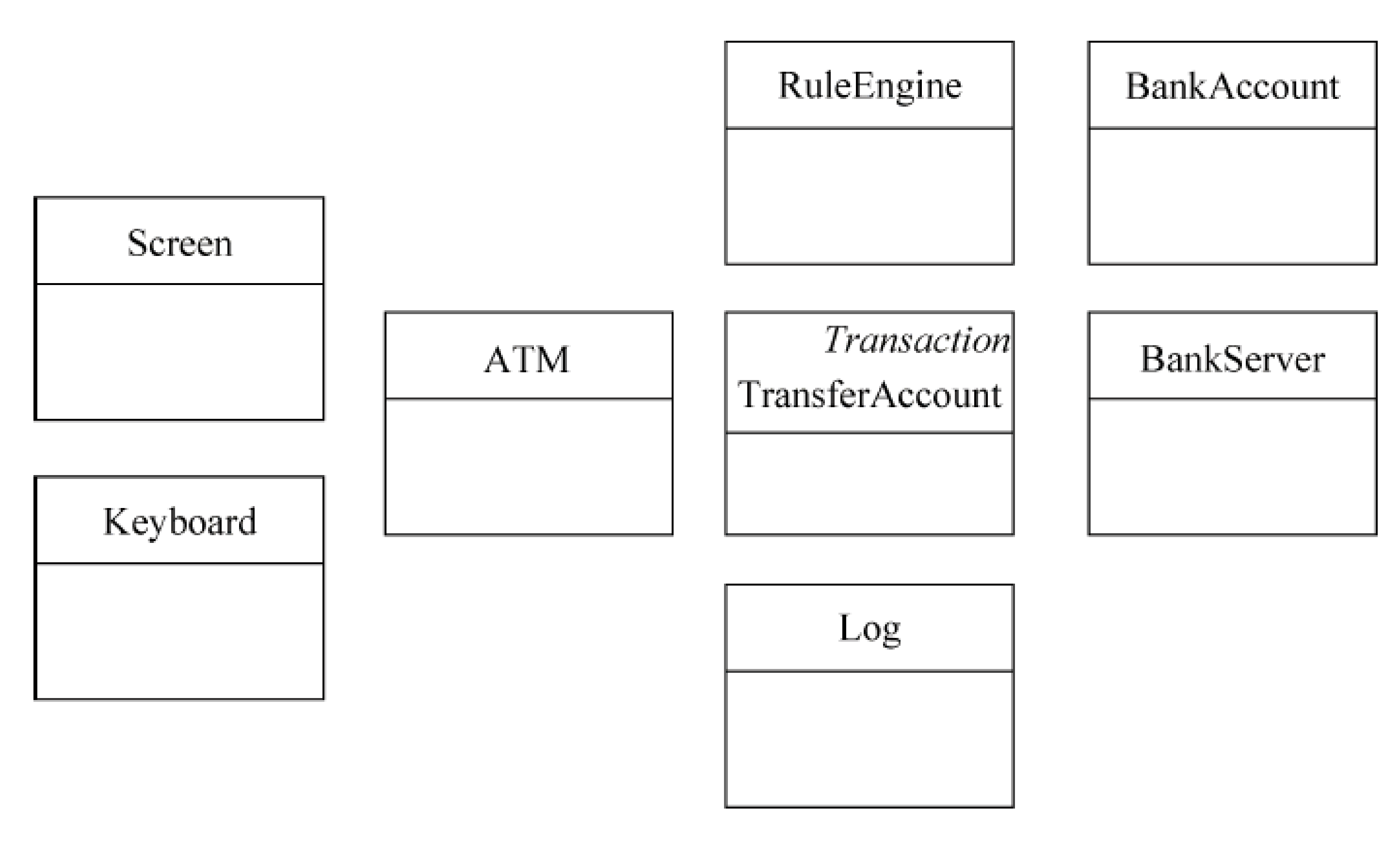


图 12-24　参与转账用例协作的类

在实际中用户的输入是通过键盘设备,而 ATM 的输出是通过屏幕。为了系统设计方便,可以简化模型,选择屏幕作为用户与 ATM 机交互的媒介,负责输入和输出。因此把键盘类承担的职责转移到屏幕类。在 ATM 系统中使用账户代表客户对象,银行客户是系统的参与者。转账类(TransferAccount)是事务类(Transaction)的子类,具体的转账业务由它完成。

选择转账用例的转账成功场景 1,即是用例的基本流,根据用例规约描述过程,建立顺序图,如图 12-25 所示。

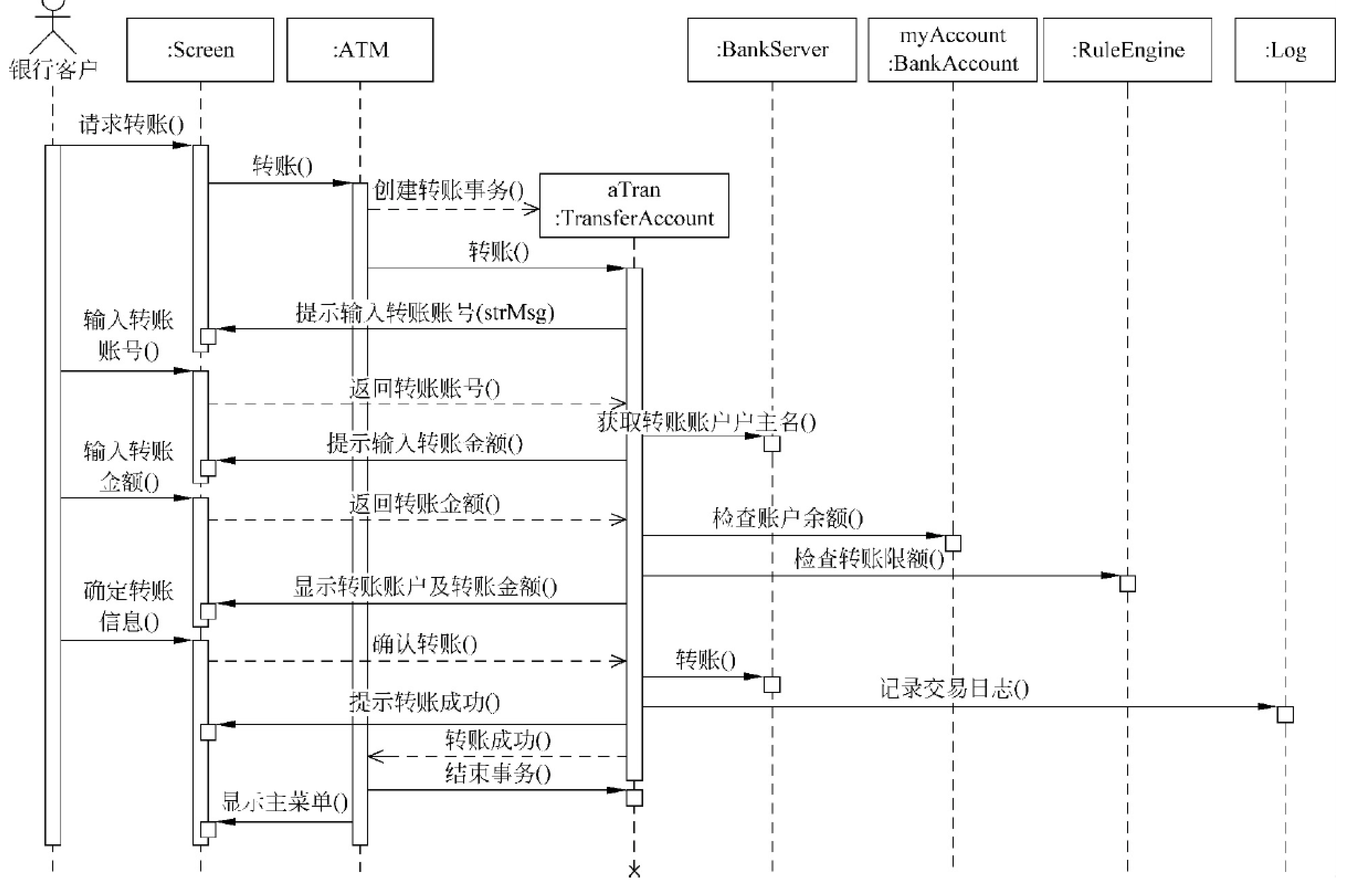


图 12-25　转账成功场景的顺序图

进行顺序图建模首先要确立建模的范围，可能是一个用例的基本流，或是备选流某个场景。建模时，通常只是把一个场景用顺序图描述出来。如果需要为用例的所有场景在一张顺序图上建模，可以考虑在协作消息上增加警戒条件，建立通用顺序图(Generic Sequence Diagram)。具体建模方法请参考《UML用户指南》①。

顺序图建模是验证用例执行逻辑及使逻辑清晰的好方法。通过观察顺序图中有什么消息发送给对象及分析被调用方法大概要花费多少时间，弄清楚如何修改设计以便均匀分配系统的职责，从而找出系统设计瓶颈。

12.4.2 协作图

对象图展示出对象和对象之间的静态关系。协作图则是对象图的扩展，除了展示对象间的关联外，还显示出对象之间的消息传递。协作图与顺序图一样，都是表示系统交互的模型，两者语义等价，可以相互转换。但协作图表示相对简洁，在对象的关联线上用箭头表示消息，加注序号表明消息的顺序。顺序图则更能表示出对象的按时间交互的整个过程，更加直观。协作图由于包含对象之间的关联线，能够阐明对象之间的关系，而顺序图则重点说明对象交互的时间顺序。当从不同角度分析问题时，需要选择合适的图。

绘制协作图时，首先要绘制一张参与协作的类的对象图，用线描述对象之间的关联，无须注明关联名，然后向图中添加对象协作的消息。图 12-26 展示了转账成功场景交互的片段。获取用户转账金额后，转账事务对象检查账户的余额，再检查转账金额有没有在规则允许的限度内，再次显示转账账户及转账金额提示用户。

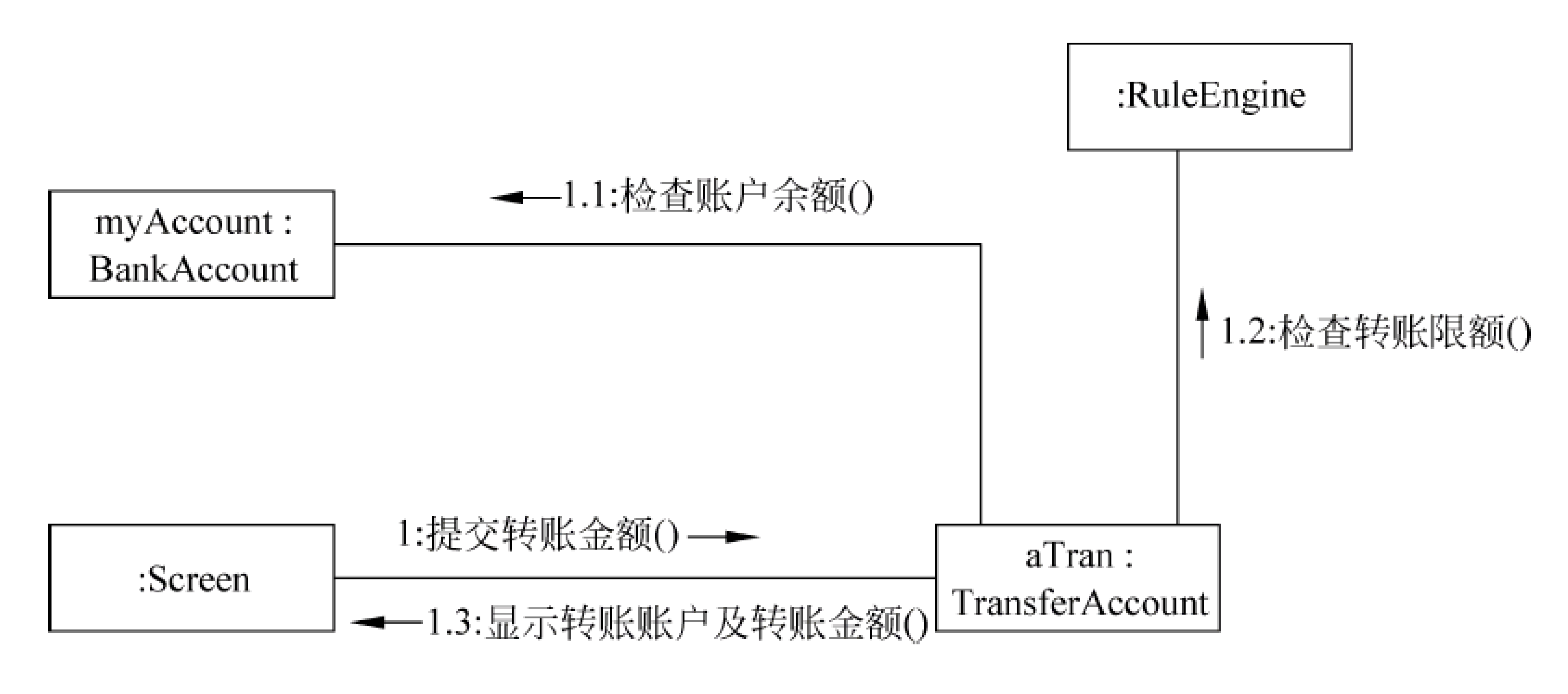


图 12-26 部分转账成功场景的协作图

12.4.3 活动图

与流程图类似，活动图表示一个业务过程中的多个顺序活动和并行活动。活动图可用于在业务单元的级别上对更高级别的业务过程进行建模，或者对低级别的内部类操作进行建模。顺序图和协作图强调的是从对象到对象的控制流程，关注的是传送消息的对象，而活动图强调的是从活动到活动的控制流，观察对象之间传送的操作。

① Grady Booch James Rumbaugh Ivar Jackson。UML用户指南. 邵维忠译. 北京：人民邮电出版社，2006

活动图是从一个表示初始活动的实心圆开始的。活动表示为一个圆角矩形(活动的名称包含在其内)。活动通过转换线段连接到其他活动,或者连接到判断点,这些判断点连接到由判断点的条件所保护的不同活动。结束过程的活动连接到一个终止点。

在 ATM 系统中,验证用户插入的银行卡,是一个相对复杂的过程,可以使用活动图描述。当用户插入银行卡时,读卡器读取卡上信息,并验证银行是否为合法的银行卡,否则直接退卡。如果是有效卡,提示用户输入密码并读取,随后验证输入的密码,如果正确进入到主菜单,如果错误则判断是否超过 3 次,超过 3 次则保留卡片,没有超过 3 次则再次提示输入密码。具体活动如图 12-27 所示。

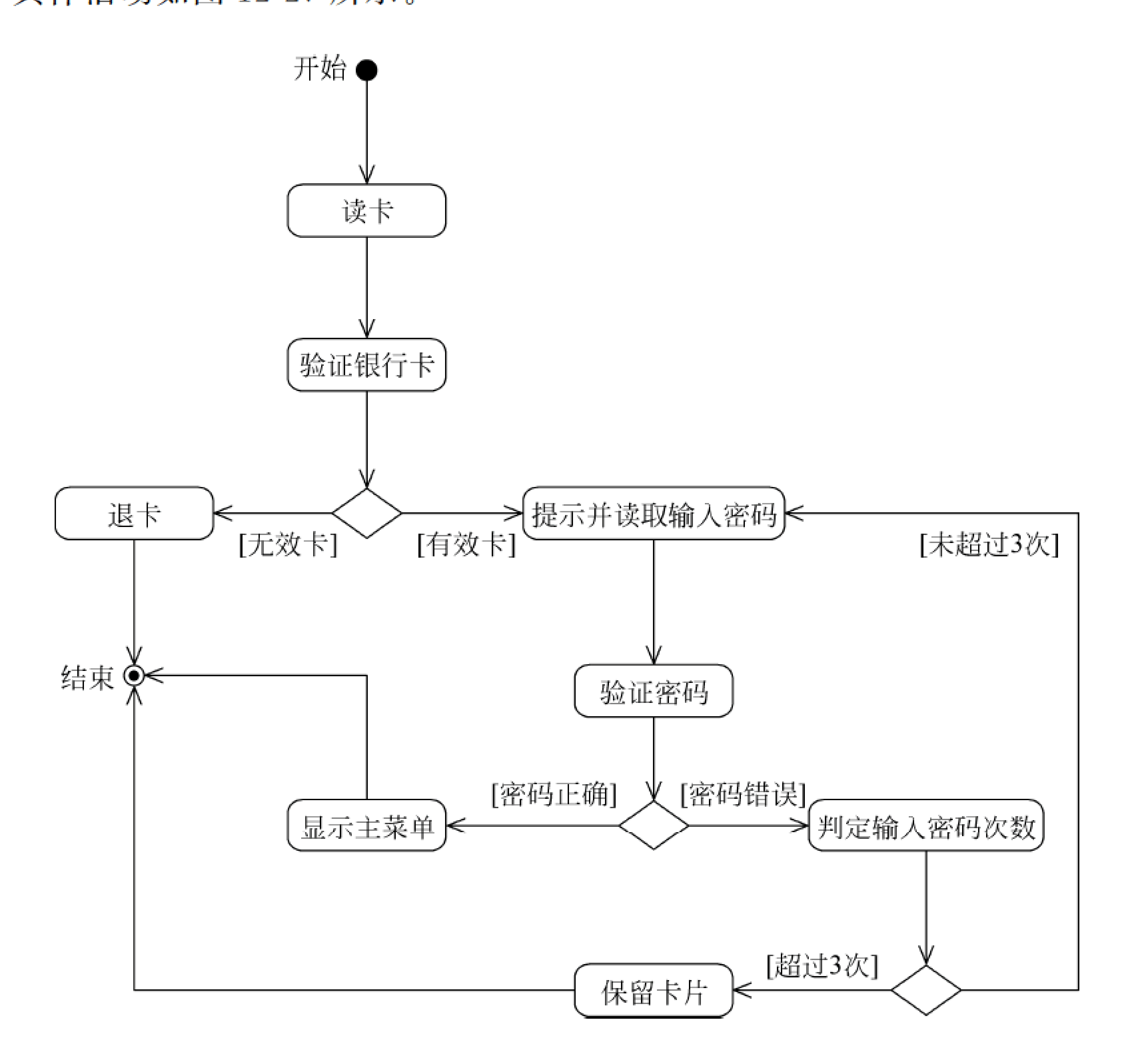


图 12-27　ATM 验证银行卡密码的活动图

活动图的一个缺点是它的扩展能力不强,当对某个业务过程建模时,并不能方便地表达出图中的各个活动分别由哪些对象负责。因此,可以将活动分组为泳道(Swim Lane),泳道用于表示实际执行活动的对象。将图用实线分割成多个平行的区域,每个区域为一个泳道,每个泳道顶部标明负责该泳道活动的对象名。

图 12-28 中显示了 ATM 取款业务的活动图,有 4 个泳道,顶端的对象控制着各自的活动。顾客输入取款金额后,ATM 判断顾客的账户余额是否足够,余额不足或者取款超过当日顾客取款限额,提示顾客修改,否则弹出钞票,同时询问用户是否打印。

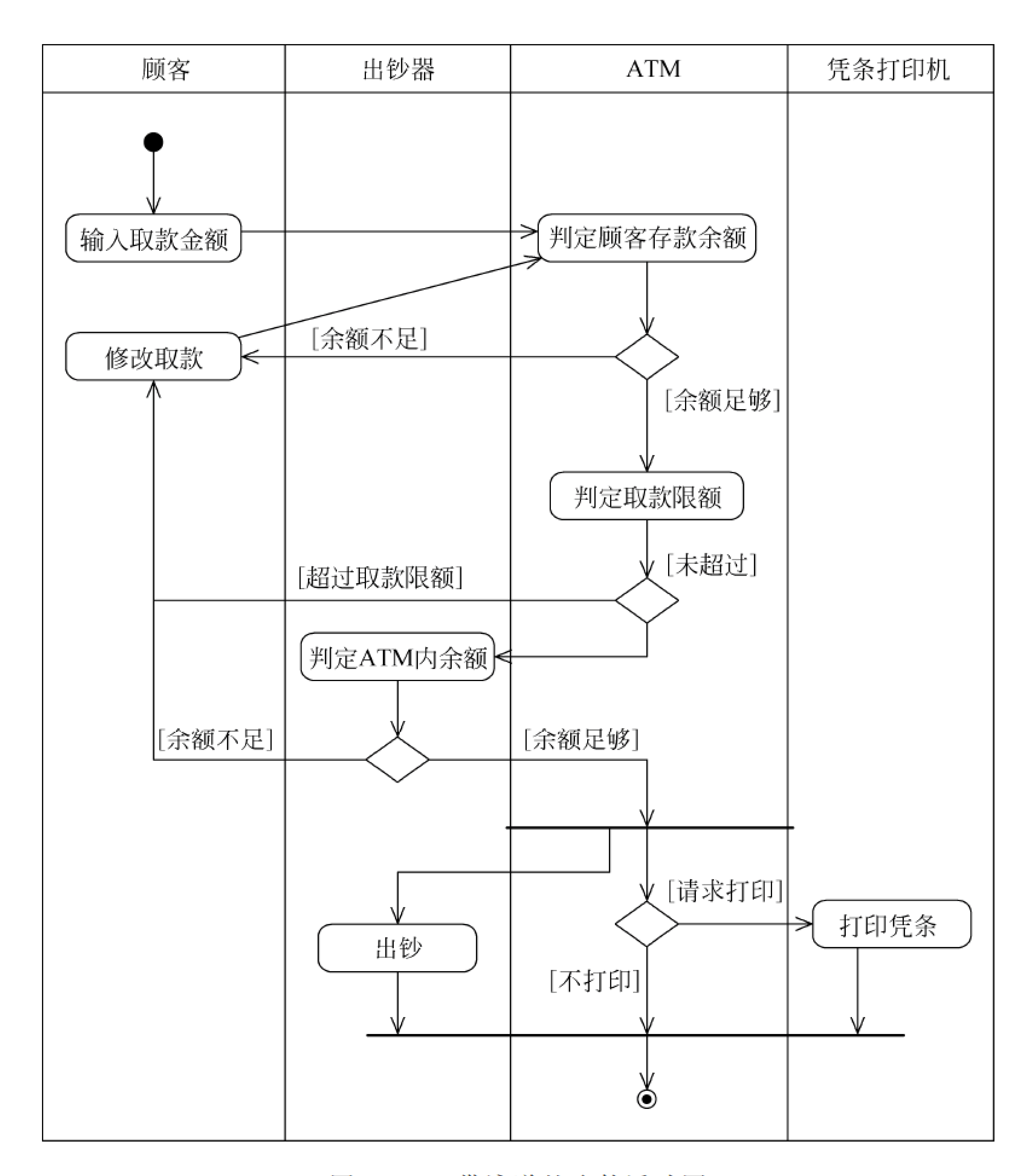


图 12-28　带泳道的取款活动图

12.5　基于控制行为建模

状态图用于系统中类的动态行为建模。大多数情况下,它包括对反应型对象的行为建模。一个反应型对象的行为是通过对来自它的语境外部的事件做出反应来刻画的。反应型对象具有清晰的生命期,它的当前行为受它的过去行为影响。活动图可以说是状态图的一种特殊情况①,状态图中大多数状态是活动状态,大部分转换都是由源状态中的活动完成触发状态改变。活动图强调从活动到活动的控制流,状态图强调对象的潜在状态和这些状态之间的转换。

每个类都有状态,但不是每个类都应该有一个状态图,只需要为依赖状态展示不同行为的类开发状态图。状态图用于描述一个对象所处的可能状态以及状态之间的转换,并给出状态变化序列的起点和终点。状态是对象处于某一表征的稳定情形,改变往往是由于响应

① Grady Booch James Rumbaugh Ivar Jackson. UML 用户指南. 邵维忠译. 北京:人民邮电出版社,2006

事件或时间流逝引起的。状态图建模，用于说明 3 个问题：对象可能处于的稳定状态；触发从状态到状态的转换的事件；当每个状态改变时发生的动作。

图 12-29 表示出了 ATM 类整个生命周期的状态图，通常用圆角矩形表示状态。ATM 的实例可能处于以下不同状态中：空闲、维护以及激活。对象从初始状态开始——用封闭的实心圆圈表示，对于 ATM 机来说没有结束状态，运行至退役。当无人使用时，ATM 机处于空闲状态，空闲状态时 ATM 机不停播放广告信息宣传企业。一旦出现故障，或是银行人员例行检查或补充钞票时，ATM 机处于维护状态，对于维护状态在此不详细分析。当顾客插入银行卡时，ATM 系统进入激活状态，开始为顾客服务。由于激活状态比较复杂，采用嵌套状态的方式表示，使整个状态图变得简洁。当 ATM 机处于激活状态时，一旦用户按了取消键或结束事务操作时，都回到空闲状态。激活状态的入口方法，即进入状态必定先执行的方法，称为 entry，是读卡操作(readCard)。出口方法，即退出状态必定执行的方法，称为 exit，是退卡操作(ejectCard)。在激活状态里，ATM 机首先进入“验证”子状态，验证卡片的有效性及用户合法性。验证通过时，进入到“选择业务”子状态，ATM 机呈现主菜单供顾客选择。顾客选择某项业务时，ATM 机进入“执行事务”子状态，执行事务时顾客请求打印则进入“打印回执”子状态，当事务执行结束顾客选择继续操作时，则 ATM 机回到“选择业务”子状态。

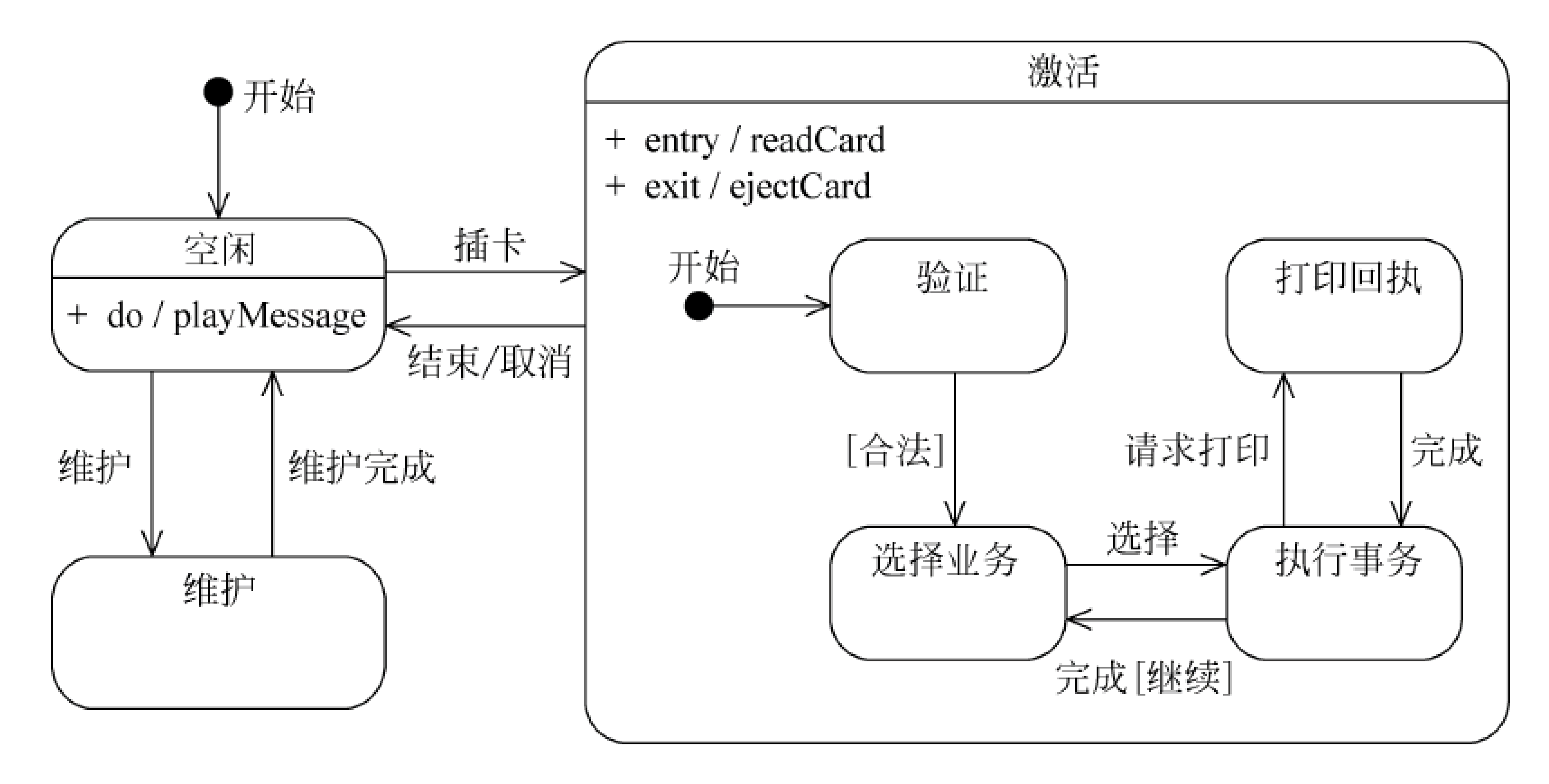


图 12-29　ATM 机的状态图

本 章 小 结

在开始构建一个面向对象系统之前，必须定义出表示待解决问题的类(对象)、类之间的相互关联和交互的方式、对象的内部结构(属性和操作)以及允许对象在一起工作的通信机制(消息)。所有这些事情均是在面向对象分析中完成的。

面向对象分析过程首先从定义用户需求开始。面向对象软件开发使用用例的方法来描述系统需求。从用户的角度来看，他们并不想了解系统的内部结构和设计，他们所关心的是系统所能提供的服务，也就是被开发出来的系统将是如何被使用的。使用用例建模是最有效的收集用户需求的技术。用例是重要的需求分析制品，用例强调了系统的活动视图。

然后使用类-职责-协作(CRC)建模技术为问题域标识参与协作的对象，为这些对象和

它们的属性与操作建立模型,同时也提供了发生在对象间的协作的初始视图。这是待构建系统的领域模型,领域模型是现实世界中对象的概念透视图,描述系统的对象组成。

接着面向对象分析过程为领域对象进一步抽象和分类,设计类模型与交互模型。顺序图、协作图和活动图是对参与者使用系统的场景逻辑建模的重要手段,描述系统的动态视图。类模型是面向对象分析与设计的支柱。类模型展示系统的类和它们的相互关系(包括继承、聚合与关联)以及类的方法和属性。类模型是在CRC模型基础上扩展而来,给出了更多细节、范围更宽,是后期编码的主要依据。

此外,对于待建系统分析,还可以使用状态图捕捉系统中反应型对象的行为建模。

思考与练习

1. 简述面向对象分析的过程,说明在整个过程需要开展哪些建模活动。

2. 为什么说"红旗"牌汽车是小汽车的实例化而引擎不是小汽车的实例化?

3. 对象和属性之间的区别是什么?

4. 本章只给出ATM系统"转账"用例规约,请参照"转账"用例规约格式,编写"取款"用例规约。

5. 用例可以帮助分析系统的需求。考虑一个计算机超市销售系统,超市中出售硬件、外部设备和软件。分析这个系统的参与者有哪些。系统的主要用例有哪些。每个用例中又有哪些场景。

6. 请为下述场景建立用例图,并简单描述图中的每个用例。

一台饮料自动售货机能提供6种不同的饮料,售货机上有6个按钮,分别对应于这6种饮料,顾客可通过按钮来选择所要的饮料。每个按钮旁边有一个指示灯,用来表明该售货机中是否还有这种饮料可售。售货机有一个投币口和找零口,用来收钱和找钱。

7. 确定下列杂货店问题中的对象及对象间的联系。

一个杂货店想使其库存管理自动化。这个杂货店具有能够记录顾客购买的所有物品和数量的销售终端。顾客服务台也有类似的终端,以处理顾客的退货。它在城市的另一个终端处理供应商发货。肉食部和农产品部也有终端用于输入由于损耗导致的损失或折扣。

8. 一个公司可以雇佣多个人,某个人在同一时刻只能为一家公司服务。每个公司只有一个总经理,总经理下有多个部门经理管理公司的雇员,公司的雇员只归一个经理管理。请为上面描述的关系建立类模型。注意捕捉类之间的关联并标明类之间的多重性和角色。

9. 请识别下列客栈问题中的类,并使用6个特征筛选出潜在类。试着使用CRC建模方法分析对象的职责和对象间的协作。

Jack在一个小镇上开了一个住宿加早餐的客栈。他们有3间客房,并需要一个系统管理房间预订并监控开支和利润。在某个顾客打电话预订住宿时,他们要查看日历。如果有空位,他们要输入顾客的名字、地址、电话号码、日期、议定的价格、信用卡号以及房间号。预订必须交一天的保证金。预订在没有保证金的情况下将保存议定的时间。如果到该日期还没有交保证金,将取消预订。

10. 解释如图12-30所示类图的含义。

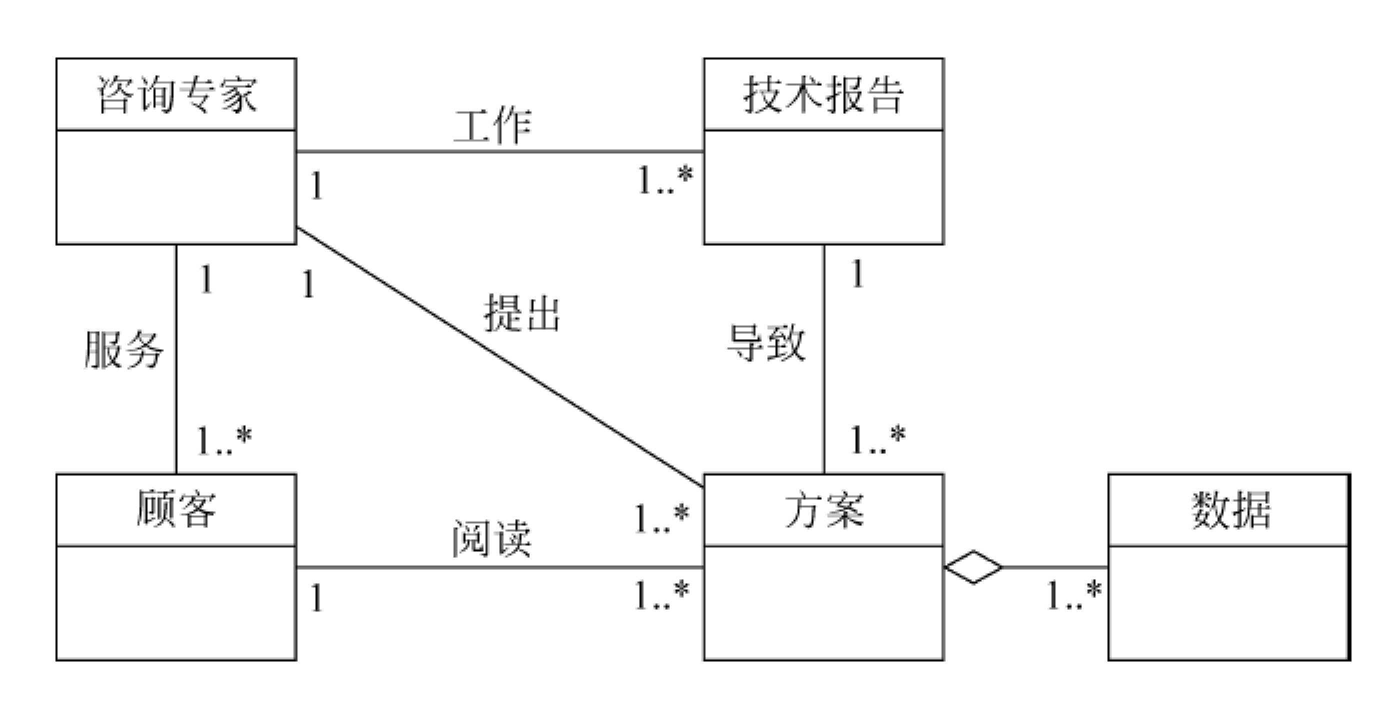


图 12-30　类图

11. 请用类图描述下列问题，要求：给出相关的类、属性、方法和类之间的关系。

一个公司雇佣了若干名员工，每个员工的信息包括员工号码、姓名、地址和生日。该公司当前有几个项目，每个项目的信息包括项目名称和开始日期。每个员工可同时分派到一个或几个项目中，也可以不做任何项目。每个项目至少由一个员工来承担。公司在每个月末给每个员工邮寄一张支票，支票上的数额与项目的性质和工作时间相关。

12. 假设增加一个新订单的流程为系统显示订单屏幕，用户必须填写订单号、日期、所订购的产品、数量、顾客姓名和地址等信息。当输入所有的信息后，用户告诉系统处理该订单。系统将把这些信息保存在数据库中并开始一个新的订单。

(1) 请识别该场景中参与协作的对象。

(2) 请用顺序图描述上述场景。

13. 请给出信用卡账户的状态图。

当一个顾客提交申请信用卡并通过时，账户处于"空"状态。一旦顾客收到信用卡并激活时，账户处于"激活但无余额"状态。当顾客用该信用卡支付时，账户处于"激活-结欠余额"状态。如果刚好支付了所有的余额，账户就处于"激活-无余额"状态。如果顾客在此1个月后还没有向银行支付消费的额度，账户就处于"拖欠账务"状态，但在信用卡允许的信用额度里仍可使用。一旦顾客支付了过去所欠的金额，账户就处于"激活-无余额"或"激活-结欠余额"状态。只有余额为0时，顾客才可以注销账户。

14. 下面给出了预订航班用例的基本流和备选流。

用例 UC1：预订航班

……

基本流(主成功场景)：

1. 顾客向系统提交航班预订信息(出发地、目的地、出发日期和人数)。
2. 系统检索满足这些条件的航班。
3. 系统显示满足条件的航班信息。
4. 顾客选择所要的航班。
5. 系统检索该航班剩余的座位信息。
6. 系统显示可预订的座位信息。
7. 顾客选择所要的座位。

8. 系统生成一个临时预订记录。
9. 系统将顾客所选择的座位状态从“未预订”修改为“已预订”。

……

备选流(扩展场景):

1a. 顾客撤销购票请求,退出用例。
2a. 系统检索不到满足条件的航班。
1. 系统向用户显示没有满足条件的航班。
2. 系统建议顾客返回步骤1。
4a. 顾客撤销购票请求,退出用例。

……

(1) 系统分析员认为与该用例相关的类至少有Reservation(预订)、Seat(座位)和Flight(航班)。请补充必要的类,建立对应于该用例主成功场景的顺序图。

(2) 请为Flight对象建立状态图。

15. Jack开设了一家汽车公司专门为学生服务,公司拥有40辆汽车,服务对象是1600个学生。汽车日常行驶的路线有30条,但在节假日等特殊日子里会临时增加新的路线。每条路线上设有许多站牌,学生们可在这些地方上下车。公司雇佣了20个全职的司机和30个兼职的司机。汽车公司设有一个调度员,专门负责司机和路线的安排。该调度员也负责将学生和家长们上下车的地点以及时间等方面的特别要求传达给司机。当路线变更或增添新路线时,调度员必须将这些信息传达给司机、学生和家长。公司经常会收到学生或家长们对司机的投诉。如果投诉的情况相当严重,司机有可能会被停职甚至被解雇。另外,公司也可能会招募新员工,以替代被解雇和退休的员工,或配备新的路线。

(1) 请给出与调度员相关的主要用例。

(2) 请用一个协作图描述从学生家长到司机的消息传递过程。

(3) 请用顺序图描述增添新路线的过程。

(4) 请用状态图描述司机的状态。

16. 请在网上搜索关于面向对象分析的文章和信息,比较并对照不同的技术,说明为什么不同方法间存在差异,并讨论不同方法的优缺点。

第13章 面向对象设计

进行软件设计有两种方式：一种是让它尽量简单，让人看不出明显的不足；另一种是弄得尽量复杂，让人看不出明显的缺陷。

——C. A. R. Hoare(图灵奖获得者)

设计的目的是为了确定如何构建系统，并且获取到足够的信息，用于驱动系统的真正实现。分析仅着眼于对将要构建内容的理解，分析模型虽然有效地确定将要构建的内容，但是却没有包含足够的信息来定义如何构建系统。

在面向对象分析阶段，针对的是现实世界的问题域，把需求转换为用OO概念建立的模型，以便于理解问题域和系统责任，最终构建一个映射问题域、满足用户需求、独立于实现的OOA模型。而面向对象设计(Object Oriented Design，OOD)就是在OOA的模型基础上运用面向对象方法，解决与实现有关的问题，产生一个符合具体实现条件的可编码实现的OOD模型，以驱动系统的实现。

OOD是以OOA模型为基础，且两者都采用UML展示，这使OOA到OOD不存在转换，只需做必须的精化和调整，进一步设计某些细节，并增加与实现相关的部分即可。因此OOA和OOD实际上没有存在很明显的界限，两个阶段是紧密衔接的。我们不必强调严格的阶段划分，这本身对系统开发毫无意义。

但是OOA和OOD有着不同的侧重点和不同的分工，OOA只针对问题域和系统责任，不考虑与实现相关的因素，建立一个独立于实现的OOA模型；OOD则考虑与实现相关的问题，建立一个具体的可实现的OOD模型。面向对象设计在开始阶段就需要做出许多技术决策：简单的问题如选用的编程语言、数据库和人机交互等，还有计划遵循的公共技术体系结构(EJB，CORBA以及Web Services等)，以及将会在何种程度上支持为系统定义的非功能需求和约束等。

对于面向对象设计主要的活动如下。

(1) 划分子系统，描述每个子系统的职责及子系统间联系。

(2) 设计系统体系结构，为系统设计适合的控制机制。

(3) 详细描述类的每个操作的过程表示和类属性的数据结构，完成对象设计。

(4) 开发对象间的交互，完成对象间的消息传递设计。

(5) 为系统开发持久模型，建立持久化机制。

这些设计步骤是迭代的，也就是说，这些步骤是增量执行的，直至最终模型的产生。

13.1 划分分析模型

对于任何大业务，把所有的业务实体和业务过程都放在一个系统中是不切实际的——结果必然过于复杂，难以使用。通过把一个大而复杂的问题分解为一系列较小而简单的问题，再分而治之，是人们日常解决问题的基本思维方式。对于一个大而复杂的系统，也要分解成若干个较小的子系统，再对每个子系统进行分析求解，最后把这些子系统集成为整个系统。这种思路与在需要分析时用包将用例模型组织起来是一个道理。在划分子系统时，可以参照用例模型分包方式进行。

系统划分出来的子系统在功能上是高内聚的，子系统中的所有类的协作都是为了达到一个共同的目标，从外部看，这些类可以看做是为了提供一个明显划定功能单元而紧密工作在一起的类。从内部看，子系统可以由类或是更小的子系统组成，这些对象相互协作，支持同一协作。

子系统只是个概念实体，可以把子系统当做大的类看待。在运行期间，它们是不存在的。子系统实现一个或多个接口，这些接口定义子系统可以执行的行为。它不能直接履行职责，需要向内部类委托完成。在 Java 语言中，通常使用包表示子系统。

子系统结构可以简化类之间的交互模式，从而降低耦合。如果没有这种子系统结构，类协作时任意一个类就可以到达任意的其他类。采用子系统封装一部分同主题的类，从而简化这些类与系统其他类的交互。

对于软件开发来说，子系统结构易于系统扩展。如当子系统重新定义新接口时，无须破坏程序的其他部分。对子系统的某一现有接口进行扩展，不会影响到系统其他部分。

因此，当设计子系统时，应该遵循下面的设计标准。

(1) 子系统应该具有定义良好的接口，通过接口和系统的其余部分通信。

(2) 除了少数的“通信类”，在某子系统中的类应该只和该子系统中的其他类协作。

(3) 子系统的数量不应太多。

(4) 子系统内部可以再次划分以降低复杂性。

通常通过寻找紧耦合的类，划分子系统。在 ATM 系统中，账户(BankAccount)和各种事务(Transaction)类协作紧密，在逻辑上这些类协作就是为了完成用户提交的金融事务，包括日志类。因此，可以把这些类看做是“ATM 金融子系统”，如图 13-1 所示。

无论是事务类还是 ATM 类都需要以各种各样的方式与用户进行交互。对与用户交互的类和设备接口类进行分组，形成一个单独的“用户交互子系统”，这样就可以为交互设计一个新的“用户交互”类，从而减少这些用户的类与事务类和 ATM 的协作，同时，有利于业务类的复用(图 13-2)。

为了划分子系统，往往会产生出新的需求。例如，为了实现系统功能就要考虑子系统间进行通信所产生的需求，而这样的需求并不是用户提出的。因此，开发时要从系统的高度对各个子系统进行设计，确定相互间的交互。对于一个负责开发子系统的小组而言，往往需要像对待整个系统一样对其建模。

ATM 系统的总体结构图通过子系统划分有了较大的改进。重新划分子系统后，改变了类之间的职责和协作，子系统模型如图 13-3 所示。划分后需要谨慎评审，确保子系统模型与原有的类模型保持一致。

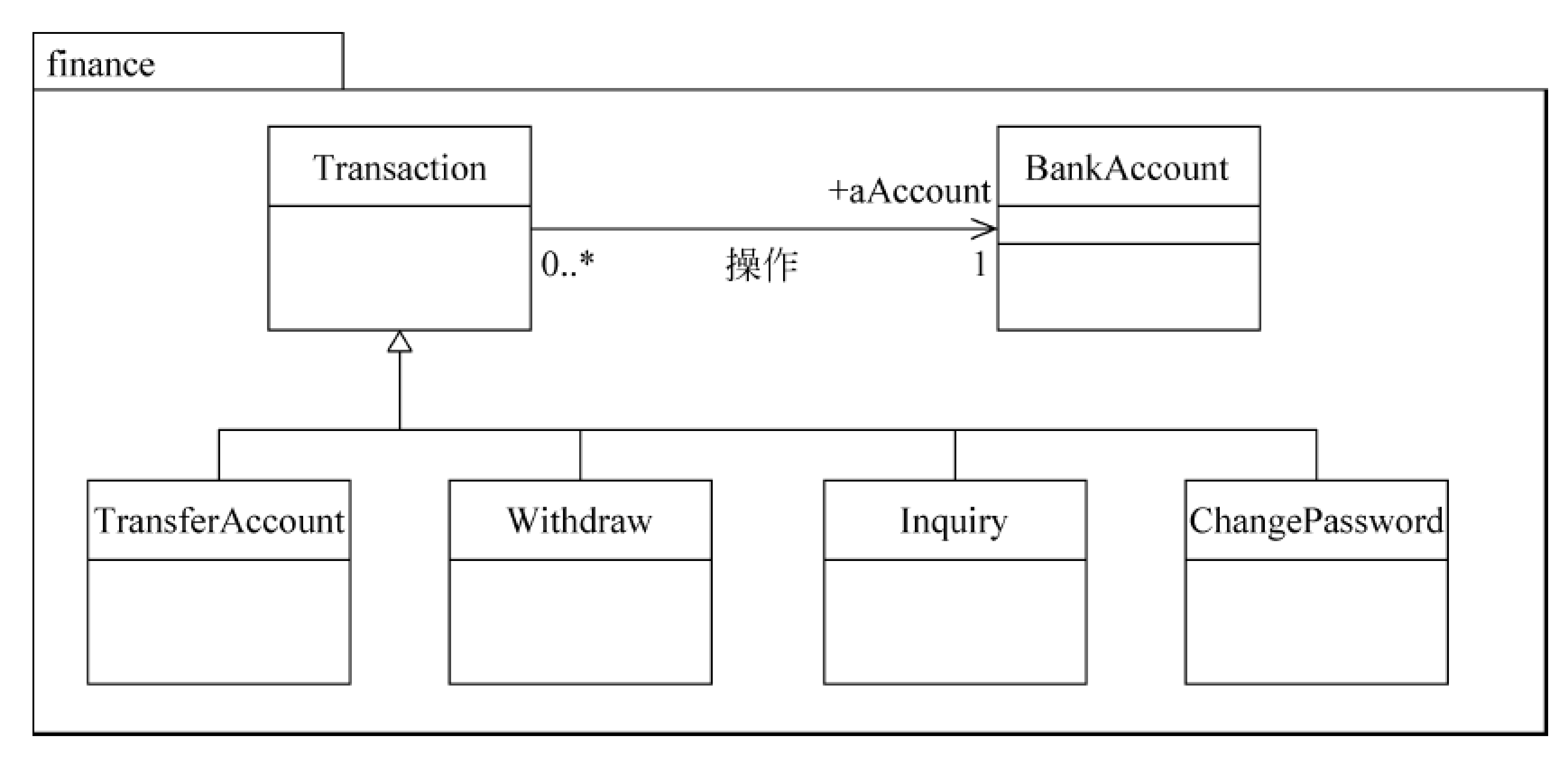


图 13-1　ATM 金融子系统

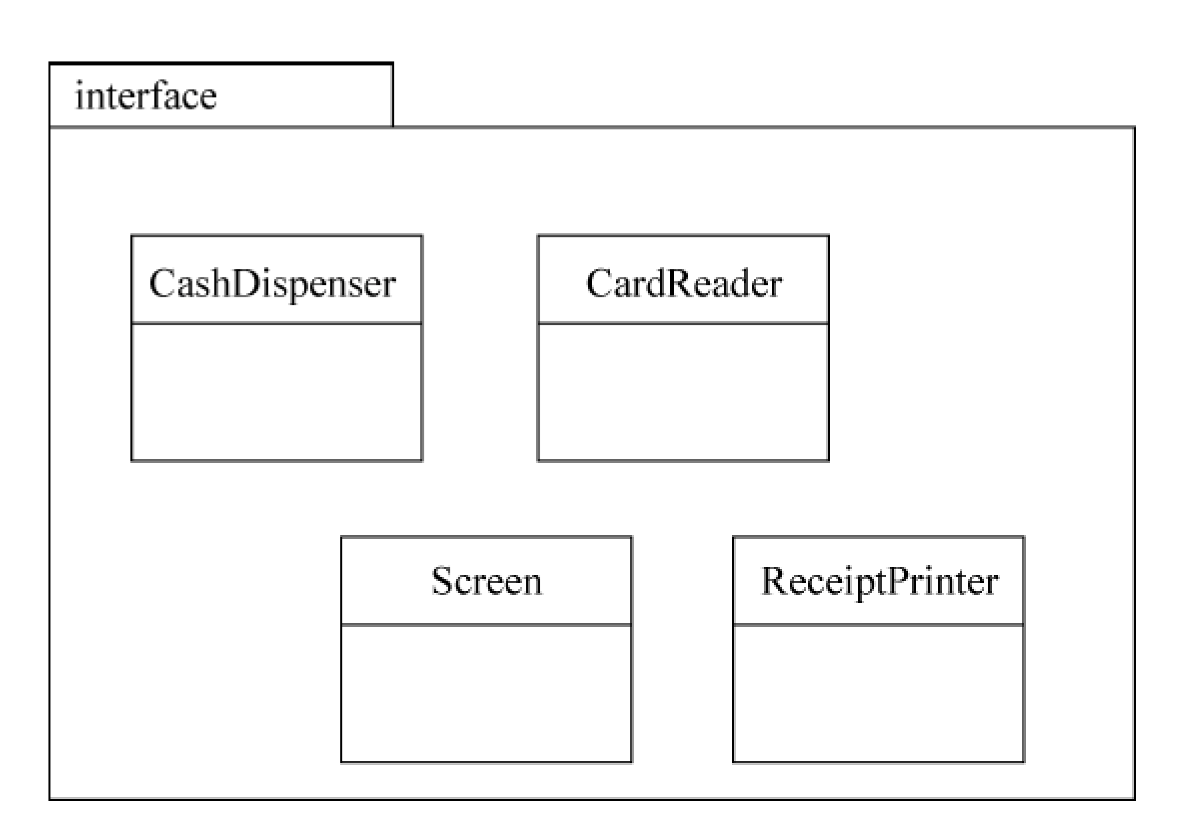


图 13-2　用户交互子系统

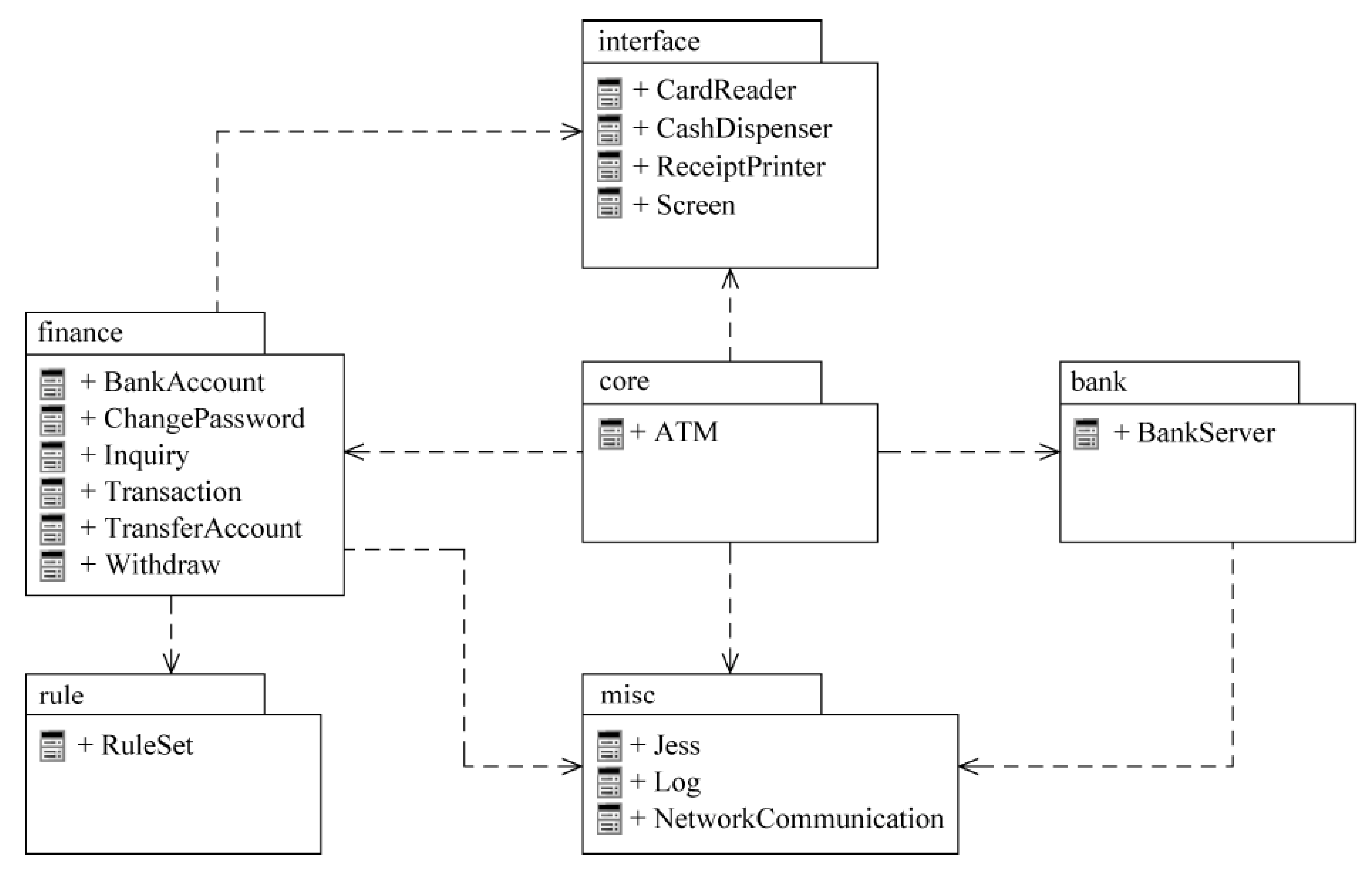


图 13-3　ATM 子系统交互

13.2 系统逻辑架构

在分解复杂的软件系统时，软件设计者用得最多的技术之一就是分层。在计算机本身的架构中，经常看到分层的例子：网络互连的协议分层实现，HTTP 层在 TCP 之上，TCP 层又在 IP 层之上；操作系统的最底层是物理层，之上是设备驱动层，系统核心运行在设备驱动层之上。

采用分层的观点来考虑系统时，可以将各个子系统组织成“多层蛋糕”的形式，每一层都在下层依托之上。上层使用下层定义的各种服务，而下层对上层一无所知。另外，每一层对自己的上层隐藏其下层的细节。将系统按照层次分解带来如下好处[①]。

(1) 每一个层次都是独立的有机整体，无须过多了解其他层次。例如，无须知道网络物理层的工作细节，照样可以使用 TCP 构建服务。

(2) 可以很容易地用新的实现来替换原有层次的实现，只要前后提供的服务相同即可。

(3) 可以降低层与层之间的依赖。如网络运营商改变了物理传输设备，但却不影响用 TCP 实现的服务。

(4) 分层有利于标准化。每个层可以定义自己的标准化，就像 TCP 是网络传输层的标准。

(5) 一旦构建好了某一层次，可以用它为很多上层服务提供支持，利于各层逻辑的复用。

(6) 如果按层分配任务，开发人员可以只关注整个结构中的其中某一层，有助于团队开发。

好的分层体系结构使系统易于扩展和维护。但是，分层架构在众多优点的背后也隐藏着如下的缺点。

(1) 层次并不能封装所有东西，有时会带来级联修改。最典型的情况就是在一个分层的设计系统中，如果在用户界面上增加一个显示的数据字段，则必须在用户界面和数据库之间的每一层进行相应的修改。

(2) 过多的层次会影响性能。由于层次的增多，同一个系统过多的跨层访问对应用程序的效率有一定的影响。

然而，分层架构中最困难的问题是决定建立哪些层次以及每一层的职责是什么。应该把握分层的原则是为了将大型逻辑结构的不同逻辑部分解耦合，对系统应适当分层，而不要滥用分层。

13.2.1 经典 3 层架构

经典的 3 层架构即将系统分为 3 层，分别是数据访问层、业务逻辑层和表现层(图 13-4)。这样分层封装的好处是分化了复杂的系统，同时也提高了系统的可维护性，也使开发过程中的分工协作更加方便快捷。

---

① Martin Fowler. 企业应用架构模式. 北京：机械工业出版社，2004

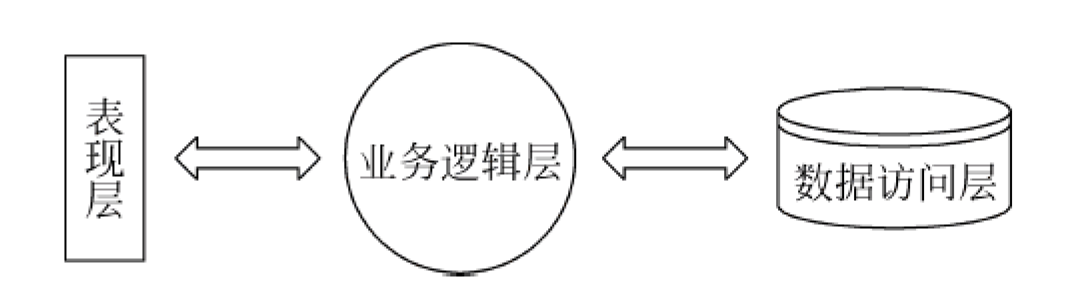


图 13-4　经典的 3 层架构

表现层即用户界面层，提供用户操作接口，表现逻辑处理用户与系统之间的交互。主要职责是向用户显示信息并把从用户那里获取的信息解释成业务逻辑层或数据访问层的各种动作。可以是简单的命令行或是文本菜单系统，也可能是功能完善的胖客户端（Rich Client），或是基于 Web 的浏览器界面。

业务逻辑层则实现对业务逻辑的封装，必须实现所有领域相关的工作：包括根据输入数据或已有数据进行计算，对表现层输入的数据进行验证，根据表现层接收的命令确定调度哪些数据。

数据访问层实现对数据库操作的封装，以隔离具体业务和数据库之间的联系，使业务逻辑层无须关心具体的数据持久化策略。

13.2.2　多层架构

应用面向对象软件设计 5 层体系结构如图 13-5 所示。用户界面层实现了系统的主要界面元素。系统的业务行为由两层来实现：业务层和控制器层。业务层实现与业务领域相关的概念，如学籍管理系统的“学生”或“学分”，着眼于业务对象数据方面的因素，加上单个对象相关的行为。另一方面，控制器层则实现业务逻辑，这些业务逻辑参与其他业务类甚至其他控制类的协作。持久访问层把持久存储、检索和删除对象的能力封装起来，使底层的存储技术不暴露出来。最后，系统类为应用提供支持性技术服务常用对象和子系统，如数据库接口或错误日志。这些服务通常独立于应用，可在多个系统中复用。

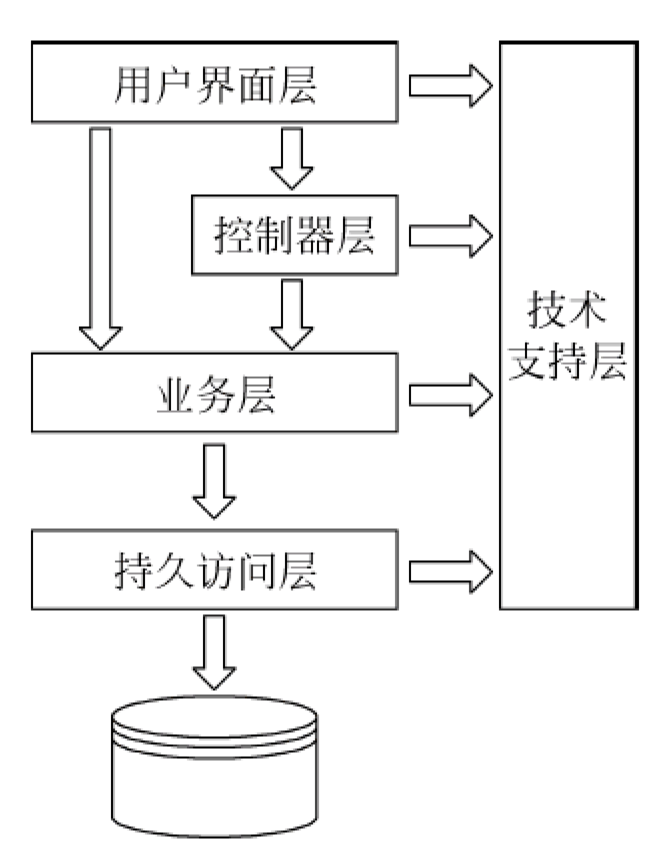


图 13-5　5 层体系结构

同一层中允许类之间的协作。如用户界面层中的类能够把消息发送到其他界面类，类间的协作还可以发生在由箭头相连的层间。如图 13-5 所示，界面类可以把消息发送到业务类，但不能直接发送到持久访问层。通过将消息的流动限制在一个方向上，就减少了类与类之间的耦合，从而大大增加了系统的可移植性。

分层并不是越多越好，过多的层次除了会给系统带来不必要的复杂性外，还会影响系统的结构设计。在业务领域对象采取分层策略，会给面向对象系统的设计带来很多麻烦。

分层时应注意以下几个方面。

（1）采用面向对象建模方法建立的业务层，应把业务逻辑都转移到业务层中，尽量多处理业务逻辑，其他层尽可能的简单一些。

（2）用户界面层尽可能与业务层靠近一点，中间不要经过太多中转，减少与用户的物理边界。

(3) 业务对象只在业务逻辑层中体现,只有在该层才可以处理对象,而在其他的层存在的只是使用类封装的数据对象,并不存在业务对象的概念。

(4) 如果一定存在分布式处理,应该通过接口访问业务对象。

(5) 不需要过度设计,分层应尽可能简洁。

在进行面向对象分析时,主要分析的是系统的业务领域对象,建立业务层的初始模型。在设计时,除了需要对业务层的模型精化外,还需要考虑到其他各层的实现,以及层间的通信。

由于 ATM 系统相对较小,系统的界面与数据来源都比较单一,不需要采用分层方式处理。仅从模拟的角度出发,可以把屏幕类、凭条打印机类、读卡器、现金分配类划分为用户界面层,这些对象都是与用户交互的接口,单独一个 ATM 类作为控制器层,居中协作业务实现。金融事务类和业务规则等类组成了业务层,而像日志及其他技术辅助类属于技术支持层,后台银行接口则可以视做持久访问层。

13.3 类模型设计

虽然在面向对象分析中,已经得到了系统的类图,但是类模型关注的是问题领域,而设计阶段的类设计,其关注焦点在于系统具体实现解决方案的领域。设计的目的是基于实现的技术变更类模型。例如,可能使用独立业务规则引擎实现业务规则,业务类访问引擎获取规则,而不是在方法中实现业务规则,增加业务规则变化的灵活性;也可能应用一些设计模式提高模型的设计质量。类模型的设计反映了设计者做出的技术决策。

13.3.1 类的设计

分析模型定义了一组完整的分析类。这些类关注用户业务领域的问题,代表问题域中的某些元素。分析类不考虑类实现的细节,抽象级别相对较高。在面向对象设计演化类模型时,主要任务是:①通过提供设计细节精化分析类,如类的属性和方法的精化,这些设计细节最终促成类的实现;②创建一组新的设计类,该设计类实现了软件的基础设施以支持业务解决方案,如添加数据库持久化的类和界面展示的类等。设计演化过程中产生的类分为 5 种不同类型:用户接口类、业务领域类、过程类、持久类和系统类,每一种类都反映了软件体系结构的某一个层次。

用户接口类定义人机交互所必需的所有抽象。人机交互是广泛的概念,不单是指与人交互的屏幕和键盘,还有一些接口的设计类。

业务领域类通常是早期定义的分析类的精化。这些类识别实现某些业务领域元素所必需的属性和方法。

过程类实现完整管理业务域类所必需的低层业务抽象。

持久类代表将在软件执行之外持续存在的数据存储。

系统类实现软件管理和控制功能,使系统能够运行并在其计算环境内与外界通信。

在分析模型演化为设计模型时,必须分别为每种设计类开发一组完整的属性和方法。分析类使用业务领域的专业术语描述对象,设计类更多表现的是实现的技术细节,将作为实

现的指导。应时刻评审每个设计类，以确保设计类是“组织良好的”。组织良好的设计类主要有以下 4 个特征[①]。

（1）完整性和充分性。设计类应该完整地封装所有可以合理预见会存在于类中的属性和方法。

（2）简单性。设计类的相关方法应该关注于实现类的某个服务。一旦服务已经被某个方法实现，类就不应该再提供另外一个完成同一事情的方法。

（3）高内聚性。一个内聚的设计类具有小的、集中的职责集合，并且专注于使用属性和方法来实现那些职责。

（4）低耦合性。在设计模型内，设计类之间相互协作是必然的。但是协作应该保持在一个可以接受的最小范围内。如果设计模型高度耦合，那么系统就难以实现、测试，并且维护也很费力。

13.3.2　接口设计

软件接口设计类定义信息如何流入和流出系统，以及被定义为体系结构一部分的构件之间是如何通信的。接口设计有 3 个重要的元素：①用户界面；②和其他系统、设备、网络或其他信息生产者或使用者的外部接口；③各种设计构件之间的内部接口。这些接口设计元素允许软件和外部通信，并使软件体系结构内的构件之间能够进行内部通信和协作。

考虑 ATM 系统中的用户界面处理逻辑，在分析阶段，为了使问题简单，所有用户界面交互都由“屏幕”类实现，编码时却没有办法按照这种方式实现。因此设计时需要以 ATM 的用户界面部分进一步优化。ATM 的用户交互界面主要有 4 类：循环播放广告信息界面、显示提示信息界面、菜单选择界面和接收用户输入表单界面。不管哪类界面都应该有标题和提示语等信息以及提供显示界面响应用户输入的服务。因此，设计 ATM 的用户界面类如图 13-6 所示。

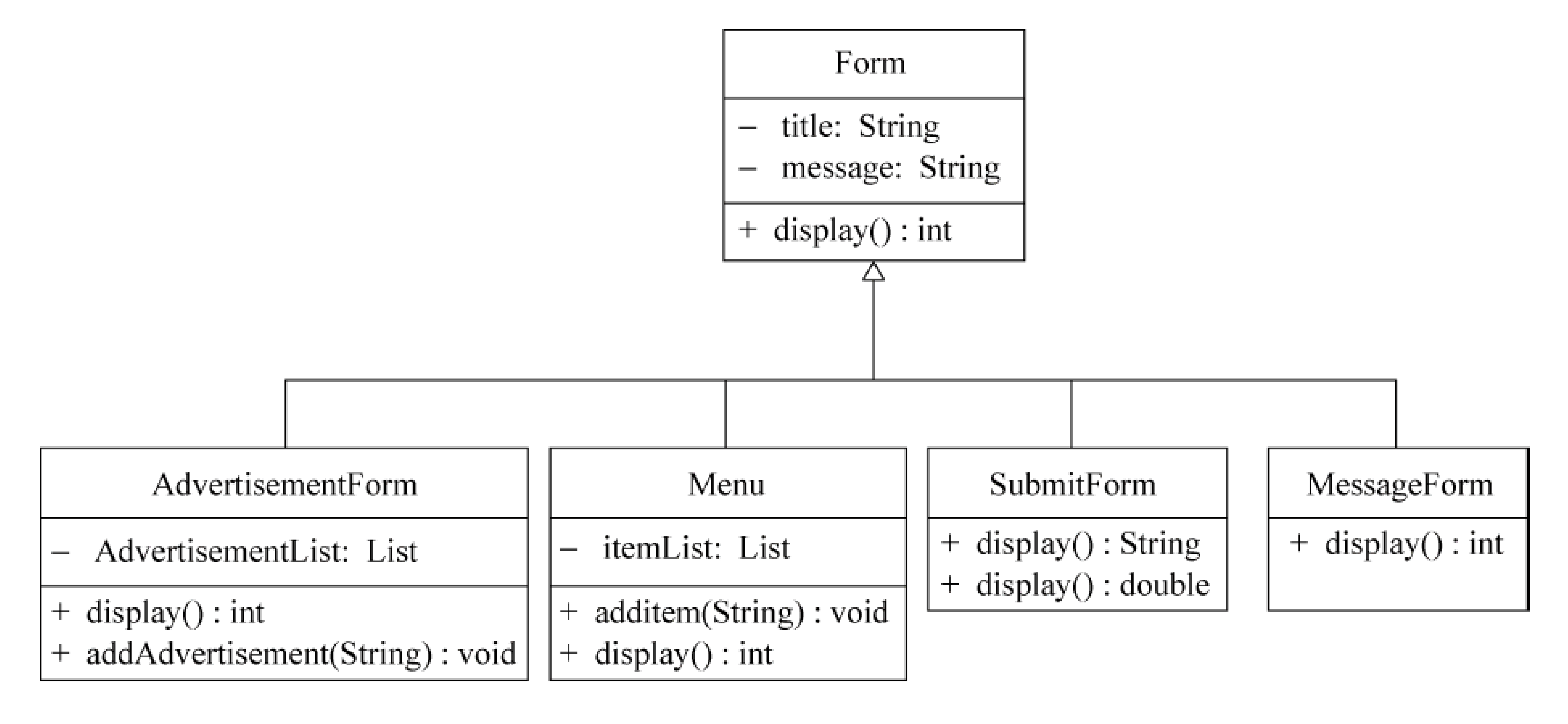


图 13-6　ATM 用户界面类分层结构

① Arlow J.，I. Neustadt. UML and the Unified Process. Addison-wesley，2002

13.3.3 属性、方法建模

在面向对象设计阶段，已经开始考虑具体的编程语言。因此，在设计类图时，要完善指明方法和属性的信息。

属性是对象数据特征。在分析模型中，仅对属性的名称进行建模。设计时，需要决定属性的类型。简单类型的属性建模不需要考虑太多，如果类型是组合数据项，如通信地址、商品列表或学习简历等，有时不需要了解组织数据项细节，可以简化处理，如将“地址”属性作为字符串类型处理。有时需要把组合数据项用另一个类描述，把组合数据项中的各项分别用新增类的属性描述。组合数据项属性调整如图 13-7 所示。

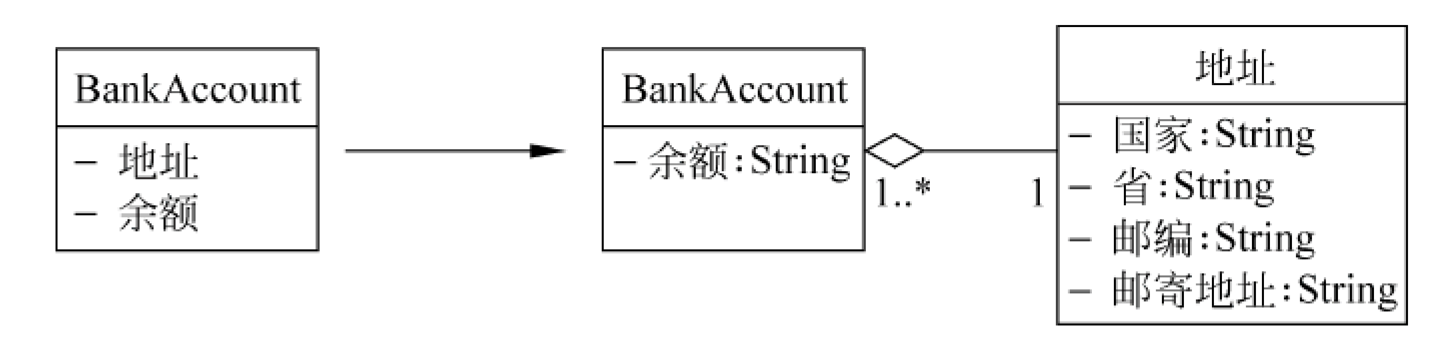


图 13-7 组合数据项属性调整

若属性有初始值或属性可取值的范围，以及对属性的一些约束，如银行账号的规则，“年龄＞0”等，设计时也都应该描述出来。

方法与结构化程序中的函数相似，是类的职责，只能通过消息访问调用方法。在设计过程中，需要指明方法的可见性，即外部对象对该方法的访问级别，还有方法的参数以及参数的类型和默认值。方法的返回值也要表示出来。对于方法实现的算法应当通过注释或文档的形式加以说明。特别是比较难懂或比较复杂的算法，必须详细记录。如果方法存在执行的错误条件或有抛出异常也都应该写入文档中。

为了实现类的封装，对属性访问通常使用 setter 和 getter 方法[①]。有时可以在 setter 方法中实现简单的逻辑验证。由于设计模型的细节已经很多，这些基本方法并不需要在类图中表示。图 13-8 是 BankAccount 类属性和方法的设计示例。

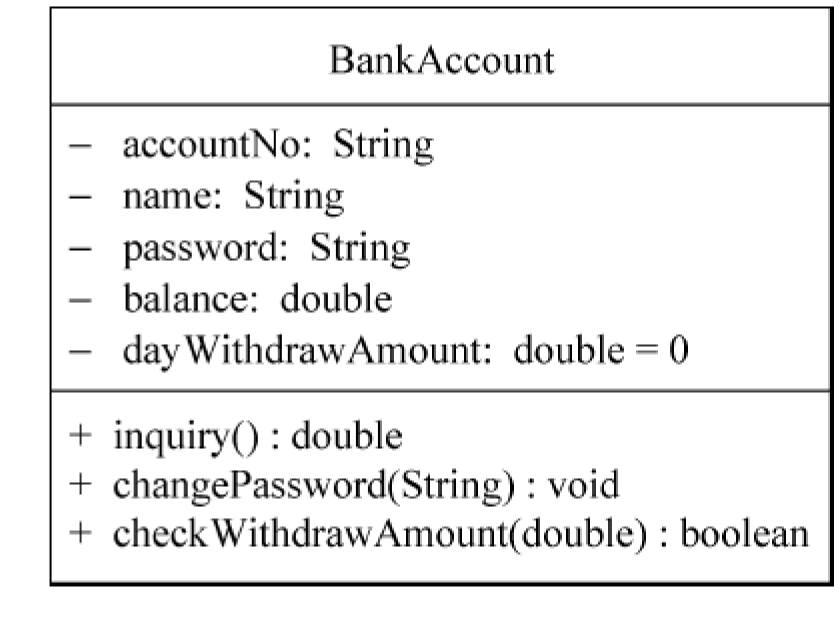


图 13-8 BankAccount 类

13.3.4 对象之间可见性设计

可见性是指一个对象看见其他对象或能够发送消息给其他对象，引用其方法的能力。在对象相互协作的过程中，顺序图描述了对象之间的消息。为了使发送消息的对象能够向接收消息的对象发送消息，发送对象必须具有接收对象的可见性，即发送对象必须拥有对接收对象的某种引用或指针。例如，“ATM”发送到“BankServer”的验证顾客身份的消息，就意味着“BankServer”对于“ATM”实例来说是可见的。当在描述用例场景实现的过程中，对象交互时，必须要保证支持消息交互的必要的可见性。

① getter 方法：获取对象的属性值的一个方法；setter 方法：设置对象的属性值的方法。

实现对象 A 到对象 B 的可见性通常有以下 4 种方式。

(1) 属性可见性：B 是 A 的属性。

(2) 参数可见性：B 是 A 中方法的参数。

(3) 局部可见性：B 是 A 中方法的局部对象。

(4) 全局可见性：B 具有某种方式的全局可见性。

在创建对象交互图时，为了使对象 A 能够向对象 B 发送消息，对于 A 而言，B 对象必须是可见的。

在 ATM 系统中，对于设备的模拟接口类，都是组成 ATM 机的一部分，模拟实现时设计成“ATM”类的属性，包括出钞器(CashDispenser)、读卡器(CardReader)、凭条打印机(ReceiptPrinter)和银行后台服务系统接口(BankServer)，因此“ATM”类对这些接口类都是属性可见性；“ATM”类与“金融事务”类，只是在执行某个金融事务时创建实例，“金融事务”类对于“ATM”类来说是局部可见性；“ATM”类需要通过参数将当前使用 ATM 机的账户信息传递给金融事务，供其操作。因此，“账户信息”对于“金融事务”类有参数可见性；而像记录日志的“Log”类，则可以定义成静态类或为其定义静态方法，提供全局可见性。

13.3.5　用例迭代实现

在设计过程中，无论是画图还是编码，对象职责的分配和协作的设计都是非常重要和具有创造性的步骤。在分析模型中，确定了用例某一个或多个场景的参与协作的对象及其实现过程。在经过类设计，明确了类的属性和方法后，需要重新迭代设计用例的实现。

以 ATM 系统的顾客“身份验证”用例为例，在用例中参与协作的有“ATM”、“CardReader”、“BankServer”和界面类，以及账户信息类。

在 ATM 空闲时，显示屏播放广告信息，为了方便模拟实现，可以将播放广告信息的界面由“CardReader”控制。具体协作过程如图 13-9 所示。图中引入边界类、控制类和实体类构造型表示，使模型信息更丰富。由“ATM”创建广告信息界面，添加好广告信息后，传递给“CardReader”类，由其显示并响应广告信息界面。

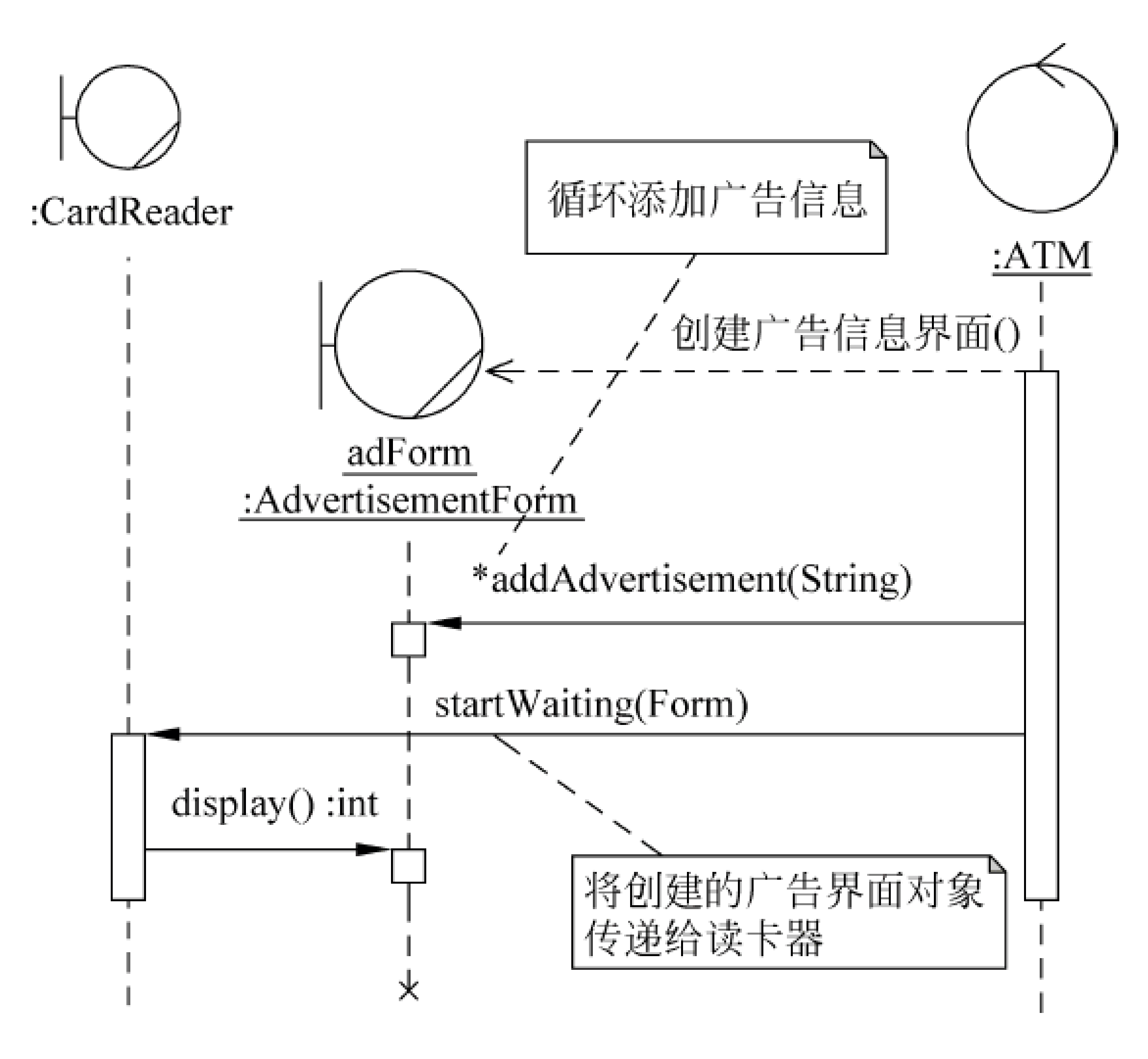


图 13-9　空闲时播放广告信息过程

当顾客插入银行卡后，读卡器首先验证银行卡的有效性，而后告知并将卡号提交给“ATM”类，“ATM”类创建“密码输入表单界面”，提示顾客输入密码。当接收到顾客输入的密码时，“ATM”向“BankServer”银行后台服务系统请求查询，当顾客提交的密码通过验证后，“ATM”请求“Log”类记录当前账户登录的信息，并显示“主菜单界面”等待顾客的下一步操作。类交互实现的过程如图 13-10 所示。

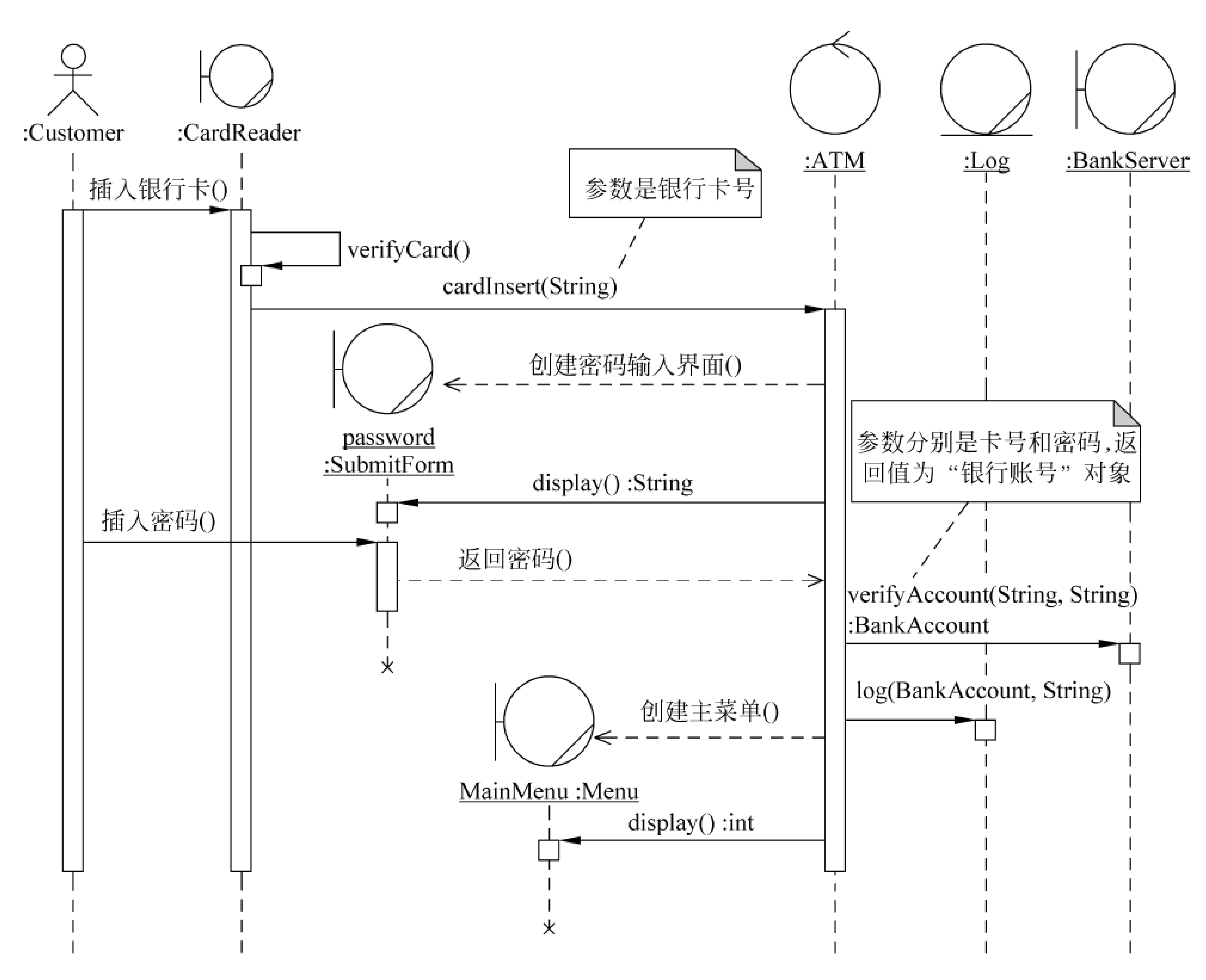


图 13-10 ATM 身份验证用例实现过程

这个用例的实现设计并不困难，但是如何合理且均匀的分配系统职责却需要多次迭代分析和设计。设计对象交互和职责分配是对象设计的核心。这些设计决策对采用面向对象技术实现的系统是否清晰、是否具有扩展性和可维护性具有重大的影响，同时也对构件复用的程度和质量具有影响。在建模时，不要试图在 UML 模型中细化所有事物，只对设计中有创造性和困难的部分进行建模。应该保持设计的轻量化和简短，快速进入编码和测试。

13.3.6 重构

重构(Refactoring)是一种重新组织系统设计的技术，可以优化构件的设计而不改变其功能或行为。Martin Fowler 首先给出了重构的定义[①]：重构是改进软件系统的过程，这种改进方式不改变代码的外部行为而改进其内部结构。即在不改变代码外在行为的前提下，

① Fowler M. ,et al. . Refacting: improving the Design of Existing Code. Addison-Wesley, 2000

对代码做出修改，以改进程序的内部结构。从本质上说，重构就是“在代码写好之后改进它的设计”。其最初起源于极限编程(Extreme Programming)过程方法中，但很快就因其完善系统的理论原则和其对软件开发所带来的极大的好处而被其他的软件开发过程所采用，并得到了软件开发人员的认可，在软件工业界也得到了良好的应用。

可能由于之前的设计存在问题：设计存在冗余性、低效或是不必要的算法、拙劣的或不恰当的数据结构等。或是可读性较差，或是效率较差。应修改这些不足以获得更好的设计。重构的每个步骤都很简单，给人“小打小闹”的感觉，但聚沙成塔，这些小的改动，累加起来，可以从根本上改善设计和代码的质量。

与重构紧密相连的另一个领域就是设计模式。虽然知道重构是为了使代码更优美，但怎样才算达到了重构的目标呢？设计模式为我们指明了方向。虽然设计模式是业界所认可的良好的软件设计结构，但不可能从软件开发的开始阶段就把软件按设计模式全面地进行设计，那样只会带来过分设计，最终浪费了大量的时间却无法获得良好的效果。而重构恰恰可以看做是软件设计的一个修正品，它可以在软件开发的过程中不断地修改现有的程序结构(即设计)，而使软件的设计朝着设计模式的方向发展，这也正是重构的目标所在。Joshua Kerievsky 这样描述重构和模式的关系[①]：模式是面向对象设计的基石，而测试优先编程和严谨的重构则是设计演进的基石。

13.4　类的设计原则

在面向对象设计中，如何通过很小的设计改变就可以适应设计需求的变化并且能减少设计修改造成的副作用，这是令设计者极为关注的问题。设计者做出技术决策时，往往需要遵循一些设计原则。许多面向对象专家提出很多类的设计原则，下面介绍 4 条有关面向对象的设计原则。

13.4.1　开闭原则

开闭原则(Open Closed Principle，OCP)指的是“一个模块在扩展性方面应该是开放的而在修改方面应该是封闭的”。这句话听起来有些自相矛盾，但却体现了一个优秀的软件设计的最重要特征。换句话来说，设计者应该采用一种直接增加代码，而无须对软件内部代码做修改就可以扩展软件功能的设计。因此为了达到这个目的，设计者在那些可能需要扩展的功能与使用该功能的类之间分离出一个缓冲区，要尽量考虑使用接口封装机制、抽象机制和多态技术进行隔离。

以电视机为例，讲述面向对象的开闭原则。收看节目时需要打开电视机电源，选择节目或进行音量调节。但是对于不同的电视机，在实现这 3 个操作的细节上往往有所不同。如果不同类型的电视机的操作都定义成不同方式，观众想使用这些不同种类的电视机时，就必须掌握这些不同类型的电视操作方式，甚至以后有新的种类电视机出现时，观众还需要重新学习操作方式，这显然对观众来说是无法接受的。因此，应统一定义一个电视机接口，提供

---

① Joshua Kerievsky. Refactoring to Patterns. Addison-Wesley Professional ，2004

开机、关机、节目递增、节目递减、增加音量、降低音量 6 个抽象方法，不同的电视机继承并实现这 6 个抽象方法。这样观众使用不同类型的电视机不用重新学习新的操作方式，而对于新的电视机扩展也极为方便。套用开闭原则的说法，对于电视机只需继承实现电视机接口定义的 6 个方法，就可实现电视机的扩展，这是“模块的扩展的开放性”。可能这种新的扩展改变了换台的实现方式，但对于观众而言使用电视机的操作方式还是不变的，“观众”类部分的代码不需要做任何修改，这是“模块修改方面的封闭”。图 13-11 是一个应用 OCP 生成的电视机类图的例子。

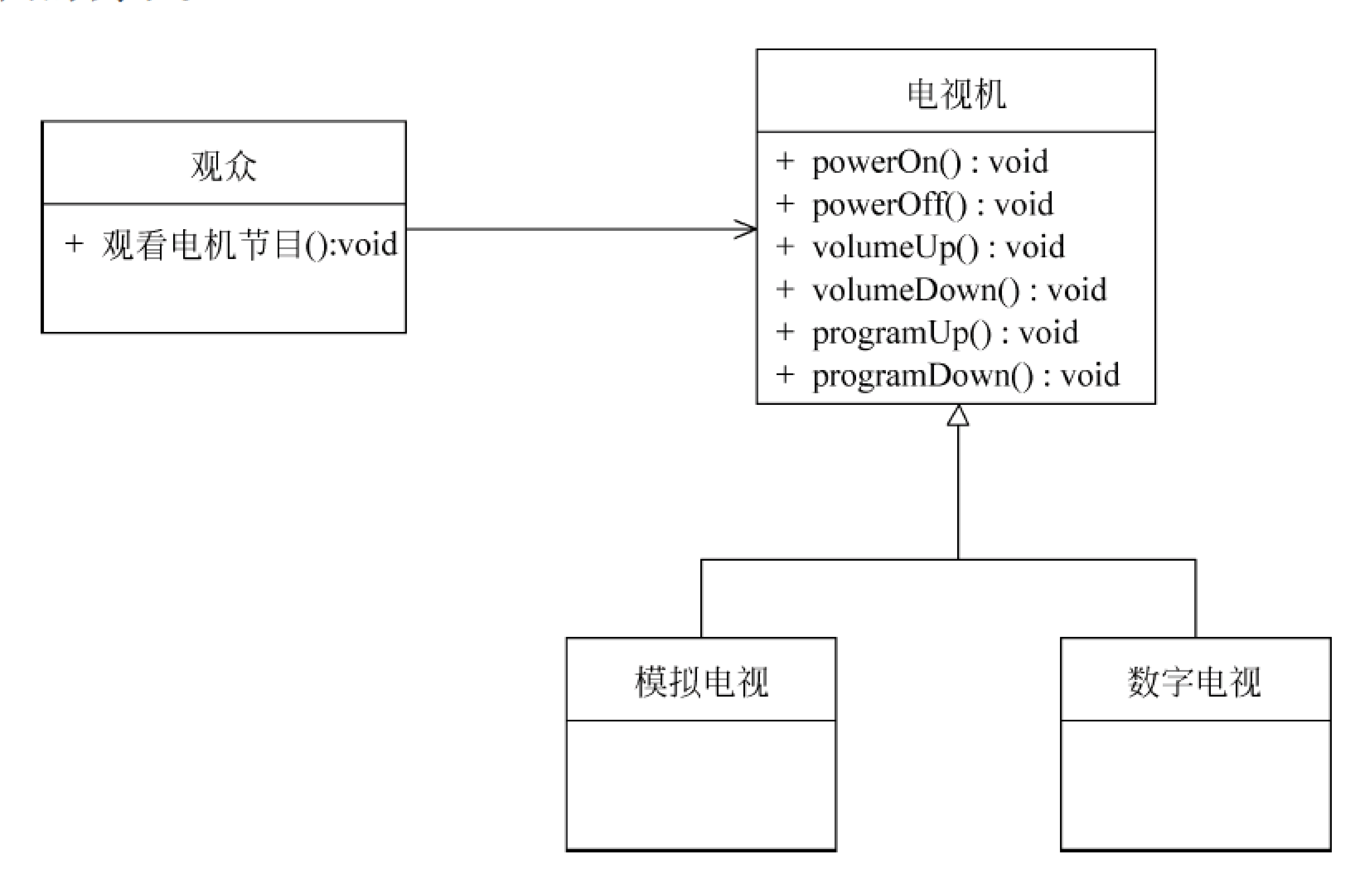


图 13-11 应用 OCP 生成的电视机类图

在 ATM 系统中，将外部设备都模拟成接口类形式，如果真实的物理设备发生改变时，如凭条打印机由针式打印变为喷墨打印，对于系统其他使用打印机的部分，却无需修改代码，实现了对修改的封闭。

13.4.2 Liskov 替换原则

替换原则（Liskov Substitution Principle，LSP）指“子类应当可以替换父类并出现在父类能够出现的任何地方”。这个原则是 Liskov[①] 于 1988 年提出的设计原则。它是为了保证继承关系的正确性，满足替换原则的继承关系，只要继承实现了基类的子类，都可以被类的使用者使用，从而实现了系统的扩展。如果建模的继承关系不能通过 Liskov 替换原则，则说明设计上可能存在问题。

在图 13-12 中，实验室所拥有的设备类型反映了实验室的类型。单独考虑每个类层次结构看起来都是很合理的。“嵌入式实验室”是一个“实验室”，“嵌入式设备”也是实验设备。然而，把它们放到一起，可能就不那么合理了。根据 Liskov 替换原理，应该能够用子类的实例替换父类，这说明能够使用设备的地方，总是能够使用该设备。根据图 13-12 所建的类模型，“实验室”由多个“设备”组成，因此“实验室”也可以由多个“嵌入式设备”组成。但是，按

① Liskov. B. Data Abstraction and Hierachy. SIGPLAN Notices，1988，23(5)

照实际情况，只有嵌入式实验室里才能有嵌入式设备。显然这样的模型某些时候会给系统带来副作用，因此需要对模型进行适当改进。让“设备”成为抽象类，并且引入“实验室设备”类。用于实现这些聚合的公共属性可能会在“设备”类中实现。“嵌入式设备”类可能包含验证代码，用于验证是否能够在当前的“实验室”中使用(图 13-13)。

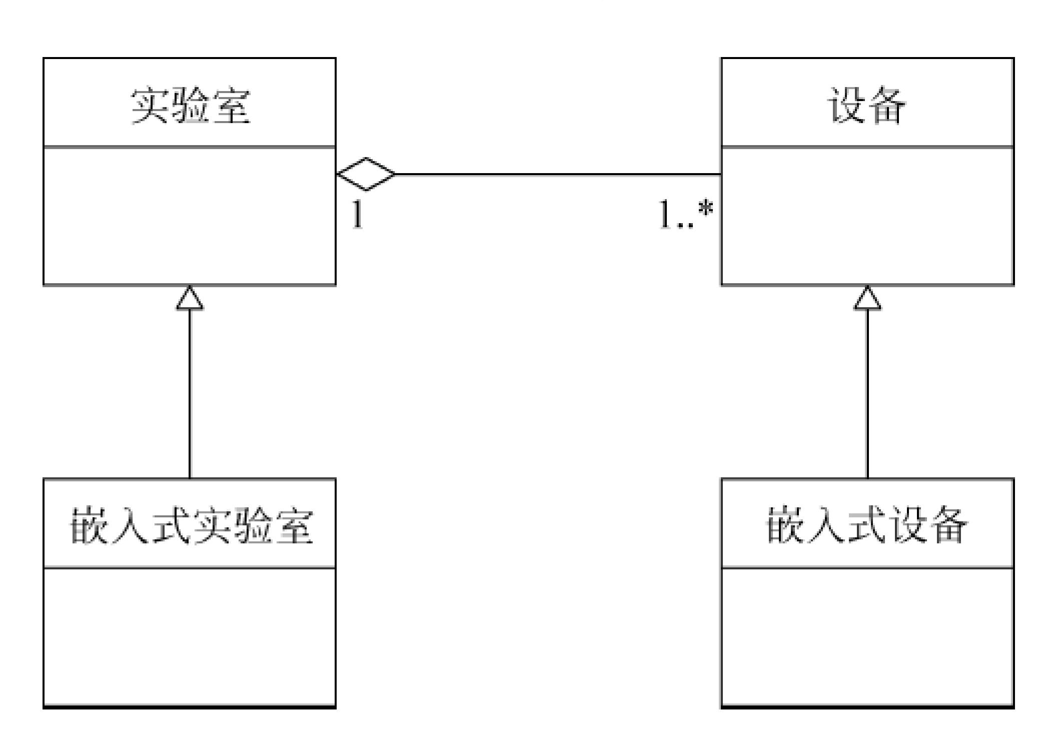


图 13-12　实验室和设备类模型

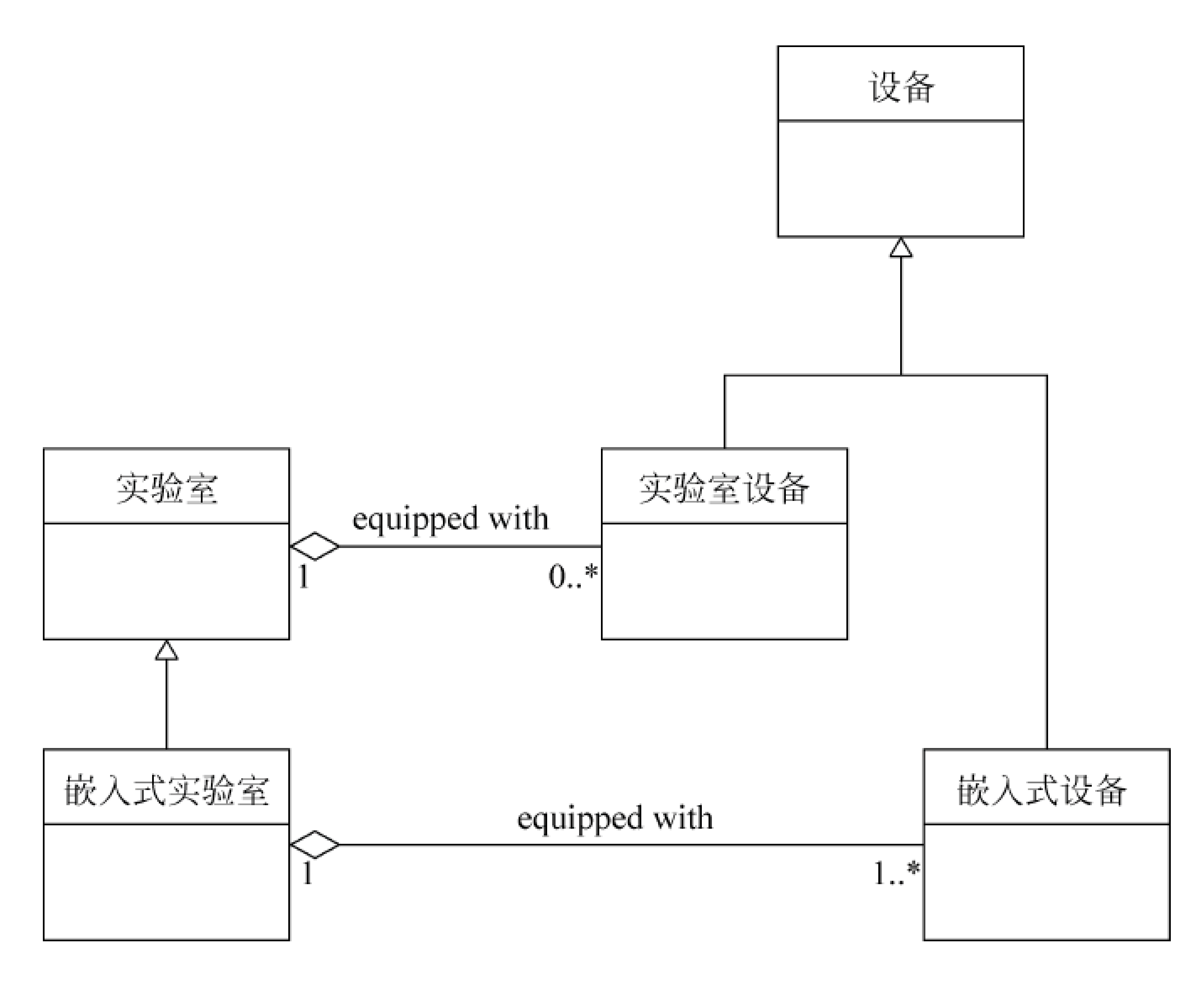


图 13-13　重构[①] (refactor)实验室和设备类模型

因此，替换原则的目的主要有以下几点。

(1) 保证系统或子系统有良好的扩展性。只有子类能够完全替换父类，才能保证系统或子系统在运行期内识别子类，因而使系统或子系统有了良好的扩展性。

(2) 实现运行期内绑定，即保证了面向对象多态性的顺利进行。这解决了大量的代码重复或冗余的问题。避免了类似 instanceof 这样的语句，或者 getClass()这样的语句，这些语句是面向对象所忌讳的。

① 通常是指在不改变代码的外部行为情况下而修改源代码。

(3) 有利于实现契约式编程。契约式编程有利于系统的分析和设计,指在分析和设计时,定义好系统的接口,然后在编码时实现这些接口即可。在父类里定义好子类需要实现的功能,而子类只要实现这些功能即可。

13.4.3 依赖倒置原则

依赖倒置原则(Dependency Inversion Principle,DIP)指"在进行业务设计时,与特定业务有关的依赖关系应该尽量依赖接口和抽象类,而不是依赖于具体类"。具体类只负责相关业务的实现,修改具体类不影响与特定业务有关的依赖关系。

为此,在进行业务设计时,应尽量在接口或抽象类中定义业务方法的原型,并通过继承实现具体的类(子类)来完成该业务方法。而业务方法内容的修改将不会影响到那些运行时调用业务方法的类。如图 13-14 所示开关面板与节能灯的例子。"开关面板"执行 toggle 方法控制"节能灯"的亮或灭。

这种方案违反了依赖倒置原则。实现时"开关面板"直接使用具体的"节能灯"对象,抽象没有和具体细节分离,当"节能灯"发生变化时,将会影响"开关面板"的实现。因此,设计一个抽象的"灯"类,而节能灯继承或实现自"灯"类。在图 13-15 中可以看到,"开关面板"与"灯"接口关联,具有更高的稳定性。"节能灯"实现了"灯"接口的方法。这样,通过倒置依赖关系的方向,"节能灯" 是依赖于其他的类,而不是被"开关面板"所依赖的。

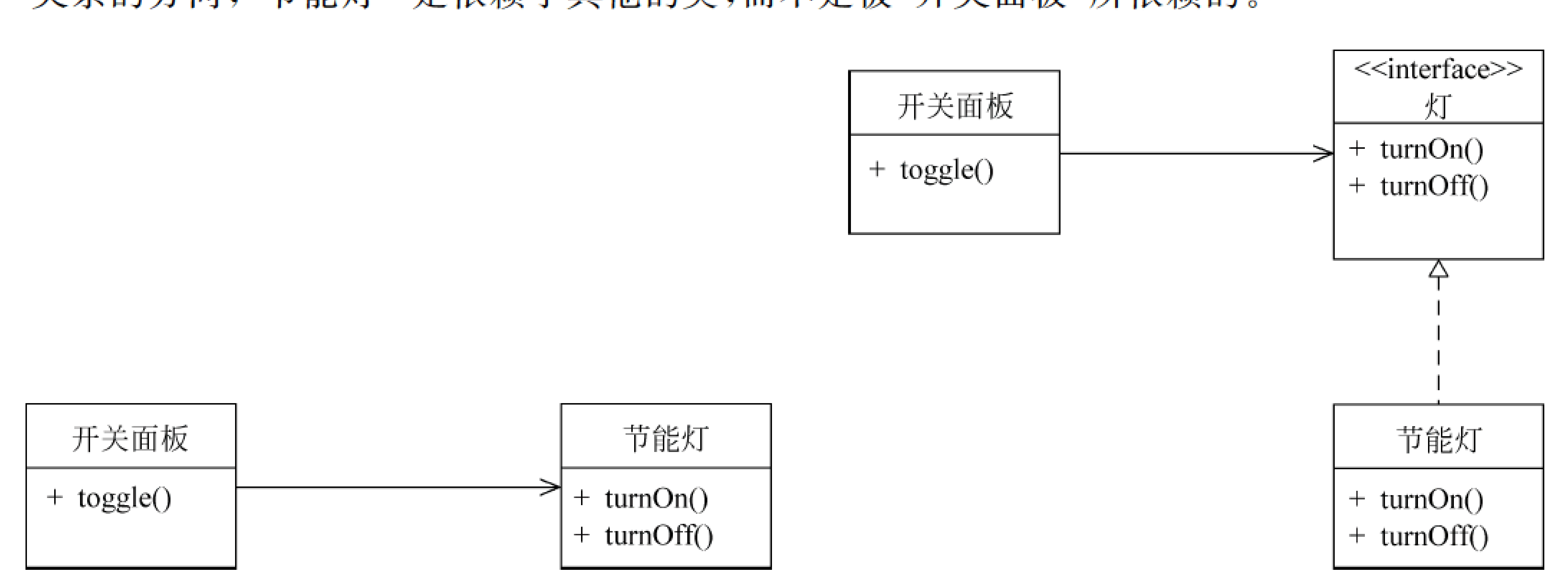


图 13-14 开关面板与节能灯的类模型　　　图 13-15 重构开关面板与节能灯的类模型

依赖倒置原则是实现许多面向对象技术优点的基本机制。它的正确应用对于创建可重用的框架来说是必须的,同时对于构建在变化面前富有弹性的代码也是非常重要的。由于抽象和细节彼此隔离,所以代码也非常容易维护。

13.4.4 接口分离原则

接口分离原则(Interface Segregation Principle,ISP)指"采用多个与特定客户类有关的接口比采用一个通用的涵盖多个业务方法的接口要好"。就一个类而言,应该只专注于做一件事和仅有一个引起它变化的原因。就是说设计这个类的功能应该只有一个,而不是两个或更多。也可以理解为引起类变化的原因,当发现要求修改这个类的原因有两个时,那么就要考虑拆分这个类了。就像一个人身兼数职,而这些事情相互关联不大,甚至有冲突,那就

无法很好地解决这些职责,应该分配给不同的人承担才对。

这个原则的本质相当简单。如果拥有一个针对多个客户的类,为每一个客户创建特定业务接口,然后使该客户类继承多个特定业务接口将比直接加载客户需要的所有方法有效。

图 13-16 展示了一个拥有多个客户的类,它通过一个巨大的接口来服务所有的客户。只要针对客户 A 的方法发生改变,客户 B 和客户 C 就会受到影响。因此可能需要进行重新编译和发布。

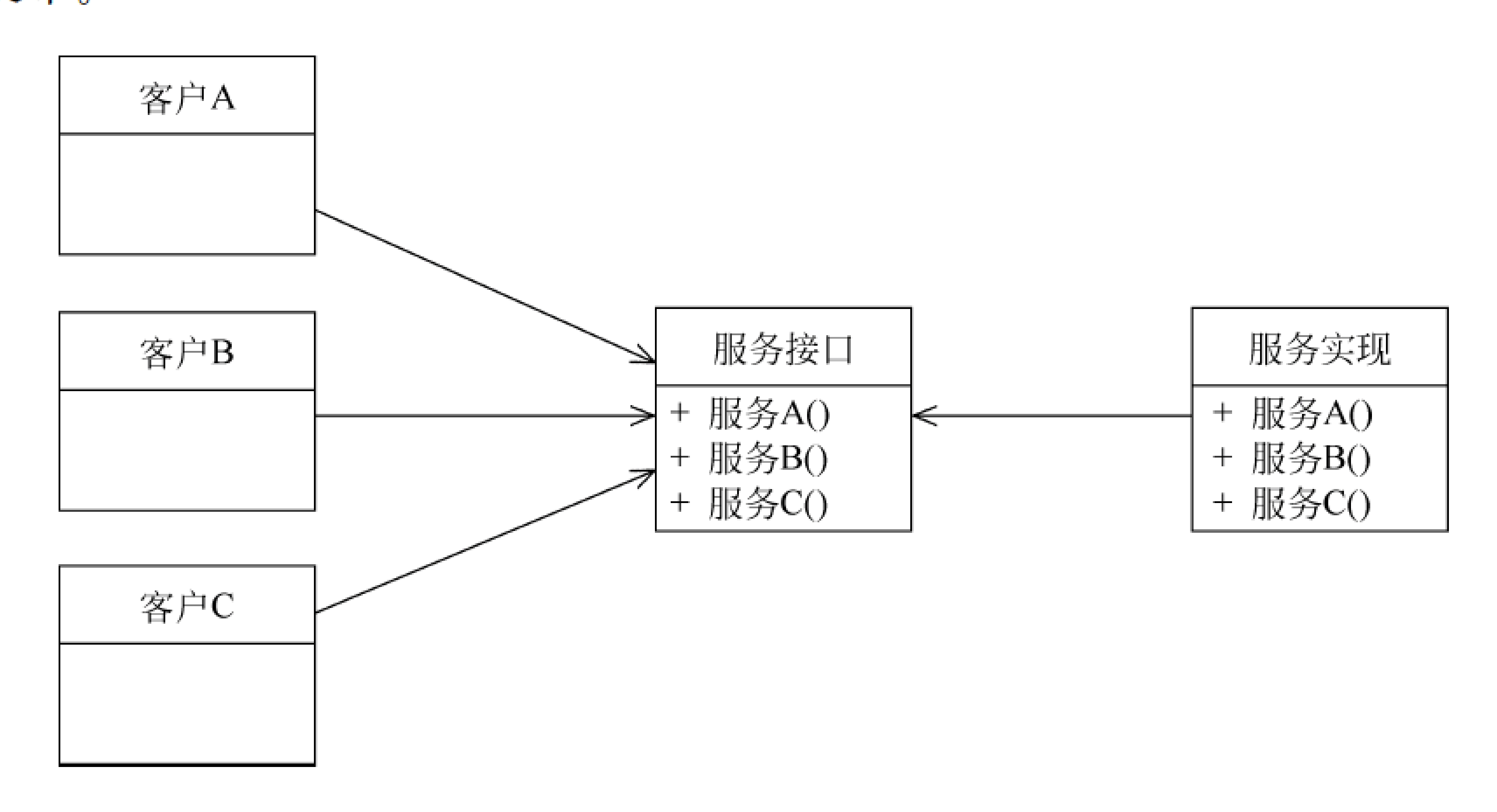


图 13-16　带有集成接口的服务类

而在图 13-17 中,每个特定客户类所需的方法被置于特定的接口中,这些接口被“服务实现”所继承并实现。如果针对客户 A 的方法发生改变,客户 B 和客户 C 并不会受到任何影响,也不需要进行再次编译和重新发布。

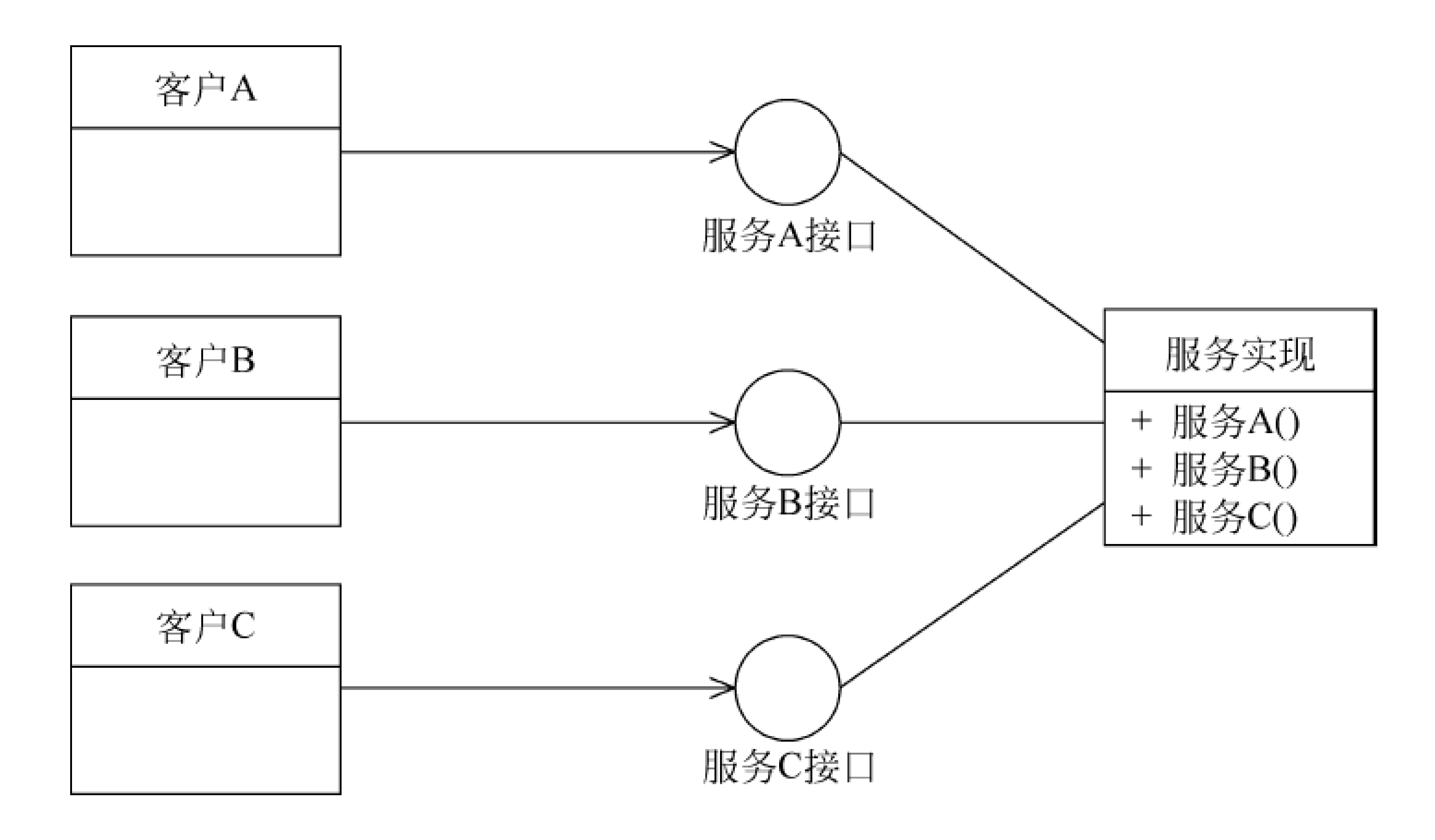


图 13-17　使用接口分离设计的类

以上 4 个原则是面向对象设计中经常用到的原则。此外,除了这 4 个原则外,还有一些常用的经验诸如单一职责原则(Single-Responsibility Principle,SRP:一个类应该有且只有一个改变的理由),迪米特法则(Law of Demeter:一个类应该对除它之外的任何事物的结构、属性和行为知道得越少越好)等可供我们在进行面向对象设计时参考。

13.5 设计模式

设计模式[①](Design Pattern)是一套被反复使用、多数人知晓的、经过分类编目的代码设计经验的总结。使用设计模式是为了可重用代码,让代码更容易被他人理解,保证代码可靠性。毫无疑问,设计模式于己于他人于系统都是多赢的,设计模式使代码编制真正工程化,设计模式是软件工程的基石,如同大厦的一块块砖石一样。

简单地讲,模式就是好的研究范例。设计模式,就是设计范例。如同在孙子兵法中,充斥着各种模式。三十六计,条条都是模式,如"远交近攻"、"走为上"、"空城计"都是各种模式的具体应用。

四人组(Gang of Four,GoF,指 Gamma, Helm, Johnson & Vlissides, Addison-Wesley 这 4 个人)的《设计模式》(1995 年出版)是第一次将设计模式提升到理论高度,并将之规范化,书中提出了 23 种基本设计模式。并非所有 23 个模式都被广泛应用,其中常用和最为有效的大概有 15 个模式。自此后,在可复用面向对象软件的发展过程中,新的大量的设计模式不断出现。

可复用面向对象软件系统现在一般划分为三大类:应用程序、工具箱和框架(Framework)。平时开发的具体软件都是应用程序;Java 的 API 属于工具箱;而框架是构成一类特定软件可复用设计的一组相互协作的类;EJB(Enterprise Java Beans)是 Java 应用于企业计算的框架。

框架通常定义了应用体系的整体结构类和对象的关系等设计参数,以便具体应用实现者能集中精力于应用本身的特定细节。框架主要记录软件应用中共同的设计决策,框架强调设计复用,因此框架设计中必然要使用设计模式。

另外,设计模式有助于对框架结构的理解,成熟的框架通常使用了多种设计模式。如果熟悉这些设计模式,毫无疑问,将能迅速掌握框架的结构。一般开发者如果突然接触某个框架会觉得特别难学,难掌握。如果转而先掌握设计模式,再来学习和使用框架无疑会事半功倍。模式不是框架,也不是过程。模式也不是简单的"问题的解决方案",因为模式必须是典型问题的解决方案,是可以让学习者举一反三的,是有研究价值、有交流价值、有自己的名字的例子。这里只介绍常用的几个设计模式,具体内容请参照 GoF 的《设计模式》一书。

13.5.1 单件模式

单件模式(Singleton Pattern)要求一个类有且仅有一个实例,并且提供了一个全局的访问点。这就提出了一个问题:如何绕过常规的构造器,提供一种机制来保证一个类只有一个实例?客户程序在调用某一个类时,它是不会考虑这个类是否只能有一个实例等问题的,所以,这应该是类设计者的责任,而不是类使用者的责任。

从另一个角度来说,单件模式其实也是一种职责型模式。因为创建了一个对象,这个对

① Erich Gamma Richard Helm Ralph Johnson John Vlissides. 设计模式:可复用面向对象软件的基础. 北京:机械工业出版社,2004

象扮演了独一无二的角色，在这个单独的对象实例中，它集中了它所属类的所有权力，同时它也肩负了行使这种权力的职责。

名称：单件模式。

意图：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

解决方案：对类定义静态方法用于返回单实例对象。

图 13-18 展示了单件模式的逻辑实现模型。其中的关键思想是，对 Singleton 类定义静态方法 getInstance，该方法提供了类 Singleton 的唯一实例。

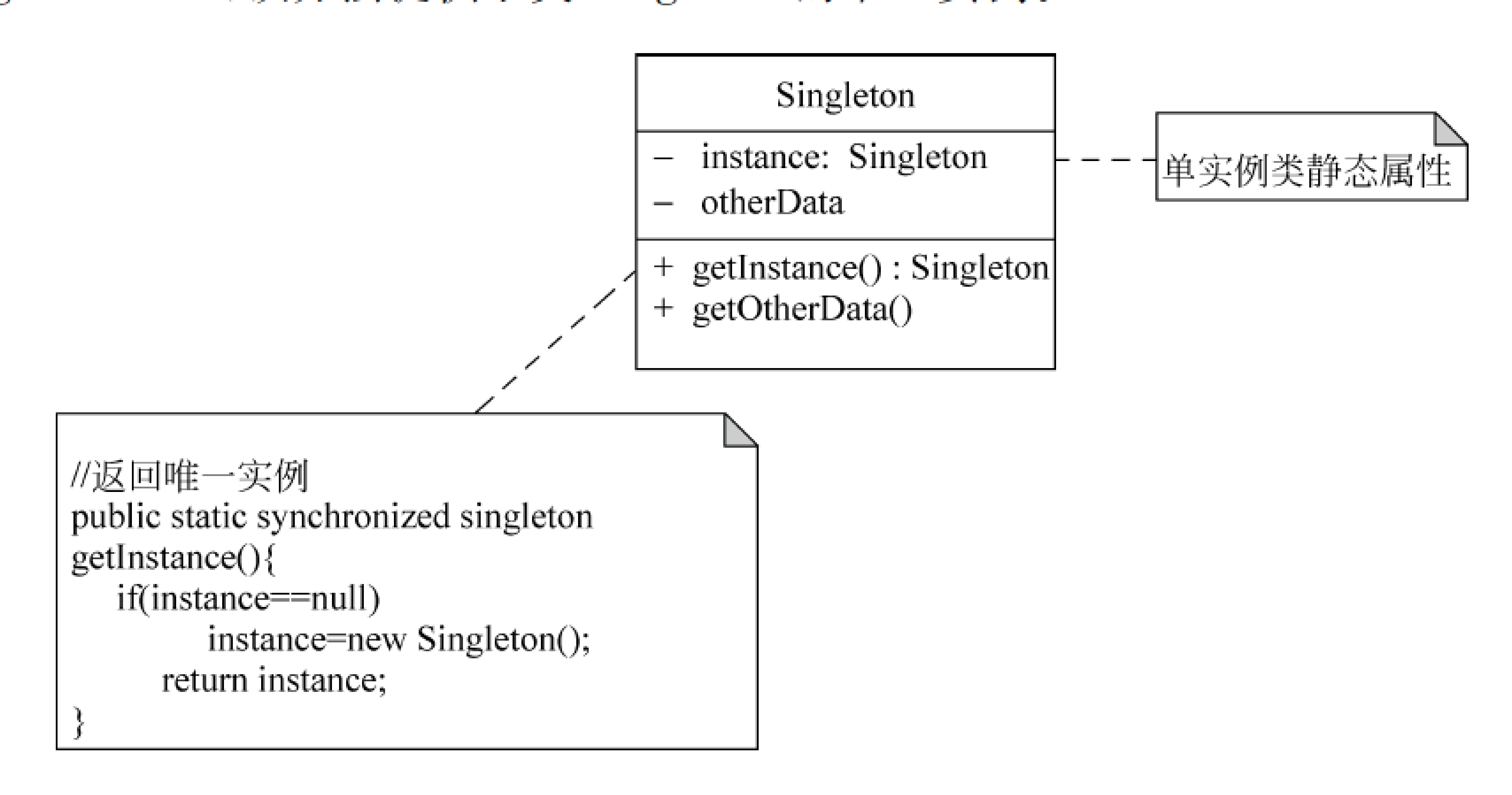


图 13-18　单件模式模型

该实现方式主要有以下 2 个优点。

(1) 由于实例是在 Instance 属性方法内部创建的，因此类可以使用附加功能（例如，对子类进行实例化），即使它可能会引入不想要的依赖性。

(2) 直到对象要求产生一个实例才执行实例化，这种方法称为“惰性实例化”。惰性实例化避免了在应用程序启动时实例化不必要的 Singleton。

单件模式是限制而不是改进类的创建。Singleton 类中的实例构造器应设置为不允许直接创建实例。单件模式只考虑了对象创建的管理，没有考虑到销毁的管理，就支持垃圾回收的平台和对象的开销来讲，一般没必要对其销毁进行特殊的管理。有时可以很简单地修改一个单件模式，使其有少数几个实例，这样做是允许的而且是有意义的。

13.5.2　抽象工厂模式

抽象工厂模式（Abstract Factory）是最一般、最抽象的工厂方法，对产品的创建细节进行了最大限度的封装，完全隔离了客户对具体产品的依赖关系。客户可以在完全对产品创建细节不知情的情况下通过更换具体工厂实现产品的更替。另外，抽象工厂模式对于产品系列的增加具有很好的支持。

抽象工厂模式适用于以下场景。

(1) 一个系统要独立于它的产品的创建、组合和表示。

(2) 一个系统要由多个产品系列中的一个来配置。

(3) 当要强调一系列相关的产品对象的设计以便进行联合使用时。

(4) 提供一个类库,而只想显示接口。

名称:抽象工厂模式。

意图:提供一个创建一系列相关或相互依赖对象的接口,而无需指定它们具体的类。

解决方案:创建名为工厂的抽象对象处理多系列产品的创建职责。

图 13-19 展示了抽象工厂模式的逻辑实现模型。Client 类依赖于抽象工厂和抽象产品实现,仅使用由 AbstractFactory 和 AbstractProduct 类声明的接口。AbstractFactory 类是一个创建抽象产品对象的操作接口,通常为每一种可以生产的产品定义一个操作。而 ProductFactory 类(即具体工厂)则实现创建具体产品对象的操作,创建出具体的产品对象。

通常在运行时创建一个 ProductFactory 类的实例。这一具体的工厂创建具有特定实现的产品对象。为创建不同的产品对象,Client 类应使用不同的具体工厂。

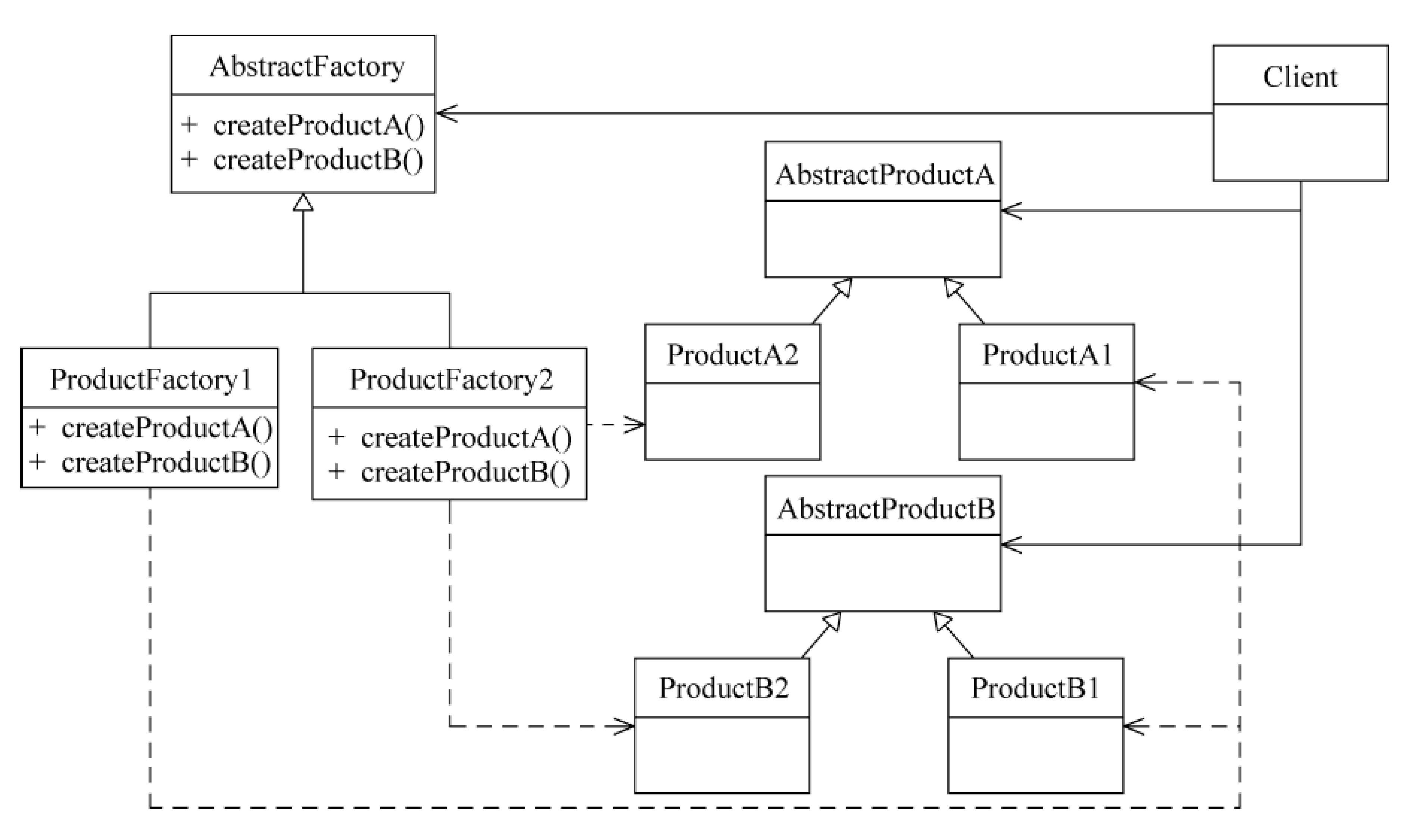


图 13-19 抽象工厂模式模型

抽象工厂模式最典型的应用是一个支持多种风格的用户界面工具包,例如 Windows 和 Motif 窗口风格。为使应用程序容易更改界面风格,可以定义一个抽象的界面组件工厂类(WidgetFactory),这个类声明了用来创建每一种风格的基本窗口组件的接口。每一类窗口组件都有一个抽象类,而具体子类则实现了不同风格的窗口组件的特定风格。对于每一个抽象窗口组件类,WidgetFactory 接口都有一个返回新窗口组件对象的操作。应用程序调用这些操作以获得窗口组件实例,但并不知道它们正在使用的是哪些具体类。这样应用程序就不依赖于具体的窗口风格组件。

抽象工厂模式将产品对象的创建延迟到它的具体工厂的子类。如果没有应对"多系列产品对象创建"的需求变化,则没有必要使用抽象工厂模式,这时使用其他的简单的静态工厂模式就完全可以。

使用抽象工厂模式的优点主要有以下几个。

(1) 分离了具体的类。抽象工厂模式帮助程序员控制一个应用创建的对象的类,因为一个工厂封装创建产品对象的责任和过程。它将客户和类的实现分离,客户通过他们的抽

象接口操纵实例，产品的类名也在具体工厂的实现中被分离，它们不出现在客户代码中。

(2) 它有利于交换产品系列。一个具体工厂类在一个应用中仅出现一次——即在它初始化时。这使改变一个应用的具体工厂变得很容易。它只需改变具体的工厂即可使用不同的产品配置，这是因为一个抽象工厂创建了一个完整的产品系列，所以整个产品系列会立刻改变。

(3) 它有利于产品的一致性。当一个系列的产品对象被设计成一起工作时，一个应用一次只能使用同一个系列中的对象，这一点很重要，而抽象工厂很容易实现这一点。

同样，使用抽象工厂模式也存在缺点，主要是难以扩展抽象工厂以生产新种类的产品。这是因为抽象工厂几乎确定了可以被创建的产品集合，支持新种类的产品就需要扩展该工厂接口，这将涉及抽象工厂类及其所有子类的改变。

13.6　对象持久性建模

应用程序运行时，总是有些数据必须被持久性存储，不管是程序运行还是休眠。持久数据存储就是即使在系统崩溃的情况下仍能存在的数据存储。持久数据存在于应用程序的活动内存之外，通常在数据库或文件系统中。虽然持久数据在系统运行时被读入内存以供使用或修改，但它最终还是要写回到外部数据存储中以长期存储。持久性建模关注如何将对象永久存储，以及对象在永久存储过程中的管理。持久对象(Persistent Object)是指需要持久性存储的对象，如银行账户类的实例。

目前采用的存储机制主要有以下 3 类。

(1) 对象数据库。使用对象数据库来存取对象，不需要其他第三方或客户定制的持久性服务。但是，目前对象数据库相对较少。

(2) 关系数据库。关系数据库很流行，应用比较广泛。如果使用关系数据库，数据的面向对象和面向记录表示之间存在失配问题。在这种情况下，需要特殊的 O-R(Object-Persistent)映射服务。

(3) 其他机制。除数据库外的其他形式，如普通文件、XML 文件等。同关系数据库一样，在对象和这些非对象化的格式之间存在失配的问题，也需要特定的服务。

大部分的数据存储都是采用关系数据库，因此这一节讨论如何将持久对象及其关联映射成相应的关系数据模型。

持久性建模由前面分析过程创建的类图驱动，通常也是迭代的过程。其中 5 个主要步骤如下。

(1) 确定数据实体。从类图中，寻找在存储数据时感兴趣的事、物或概念。

(2) 确定数据属性。对于每个数据实体，需要存储些什么信息？最容易的方法是把类的属性映射到表中某一列上。

(3) 确定数据实体的键属性。设计键属性作为数据实体记录的唯一标识符。

(4) 确定数据实体之间的关系。数据实体间的关系与类间的关系是等同的。

(5) 分解多对多关联。如果在类图中存在多对多关联，通过在持久模型中引入关联表，把多对多关联分解成两个一对多关联。

13.6.1 映射对象

进行持久性建模，首先需要分析类图中哪些是需要持久性存储的对象，将每个持久对象类在关系数据库中定义为一个表，而基本数据类型(int，String 和 boolean 等)的对象属性映射为表中的列(图 13-20)。如果对象只有基本数据类型的属性，则映射就直截了当。有时候有些属性不需要持久化存储，如收银小票类可能会有个总计的属性，可以通过其他属性计算得到。还有，对象有可能包含引用其他复杂对象的属性，需要额外考虑。

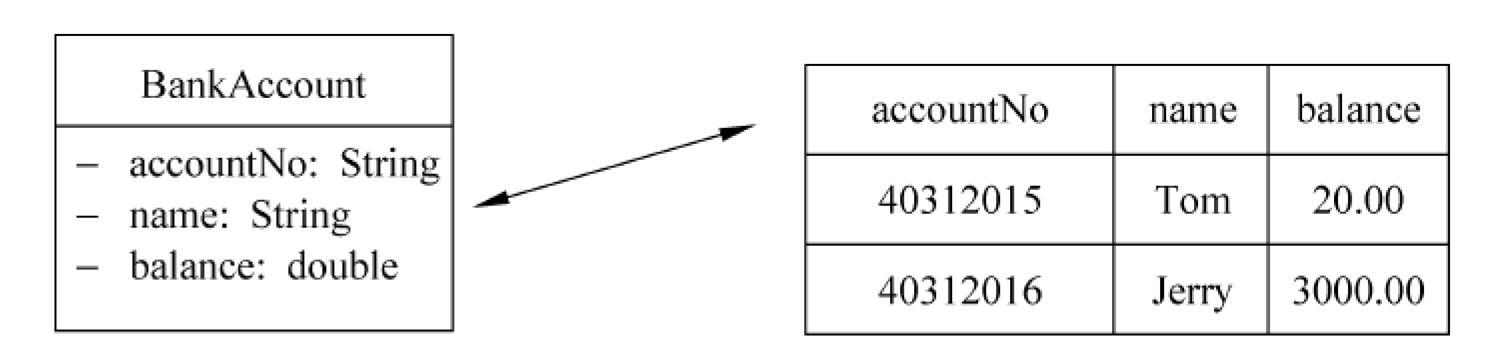


图 13-20　对象与表的映射

为了确保关系表中的记录不会导致重复对象，需要有对象和记录的一致性方法。对象标识符(Object Identifier，OID)用于在关系数据库中唯一确定对象。在关系数据库中，唯一标识符称为主键。对象标识符可以简化关系数据库中的键策略，使对象与关系表之间的关系容易维护。有很多方法可以生成唯一对象标识符，包括对于一个数据库的唯一 OID 及全局性唯一 OID。如数据库序列生成器、High-Low 键[①]生成策略等。一般使用与对象不存在逻辑关系的对象标识符，以防止由于业务规则变更而引起标识符的修改。

每个表都有一个 OID 作为主键，每个对象也有一个 OID。如果每个对象都有唯一的 OID，每个表都有 OID 主键，则每个对象都能被唯一映射到表中的某行，如图 13-21 所示。

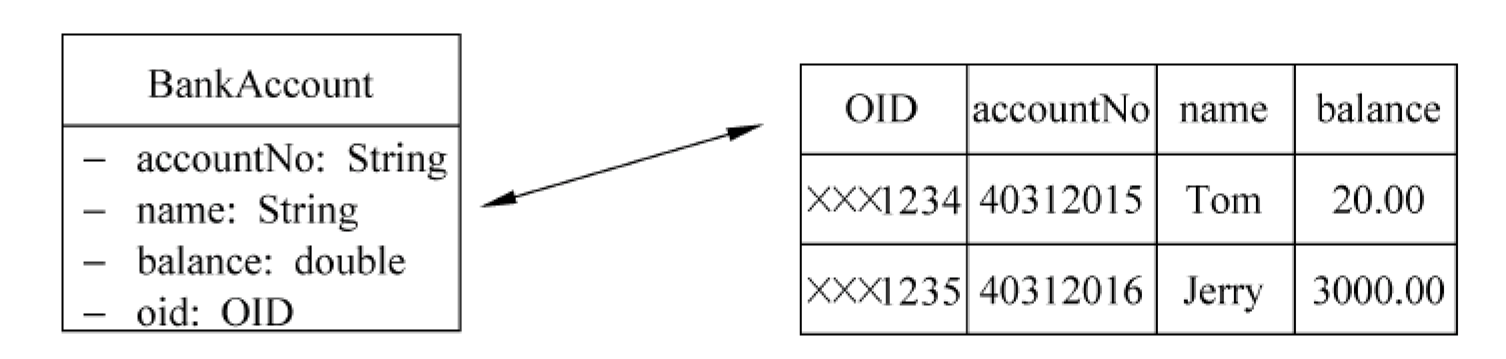


图 13-21　对象标识符连接对象与表的记录

13.6.2 继承关系映射

关系数据库不支持继承概念，当把带有继承关系的对象存进关系数据库时，需要考虑继承的映射解决方案。有以下 3 种基本的解决方法。

1. 对整个类层次使用一个数据表

把整个类层次映射到一个数据表中，类层次的所有属性都在其中进行存储。如图 13-22 所示，“经理”类和“程序员”类都继承于“职员”类。把 3 个类映射到同一张数据表中，3 个类的所有属性都映射成同一张表中的列。

---

① Ambler S. The Unified Process-Elaboration Phase. Lawrence，KA：R&D Books

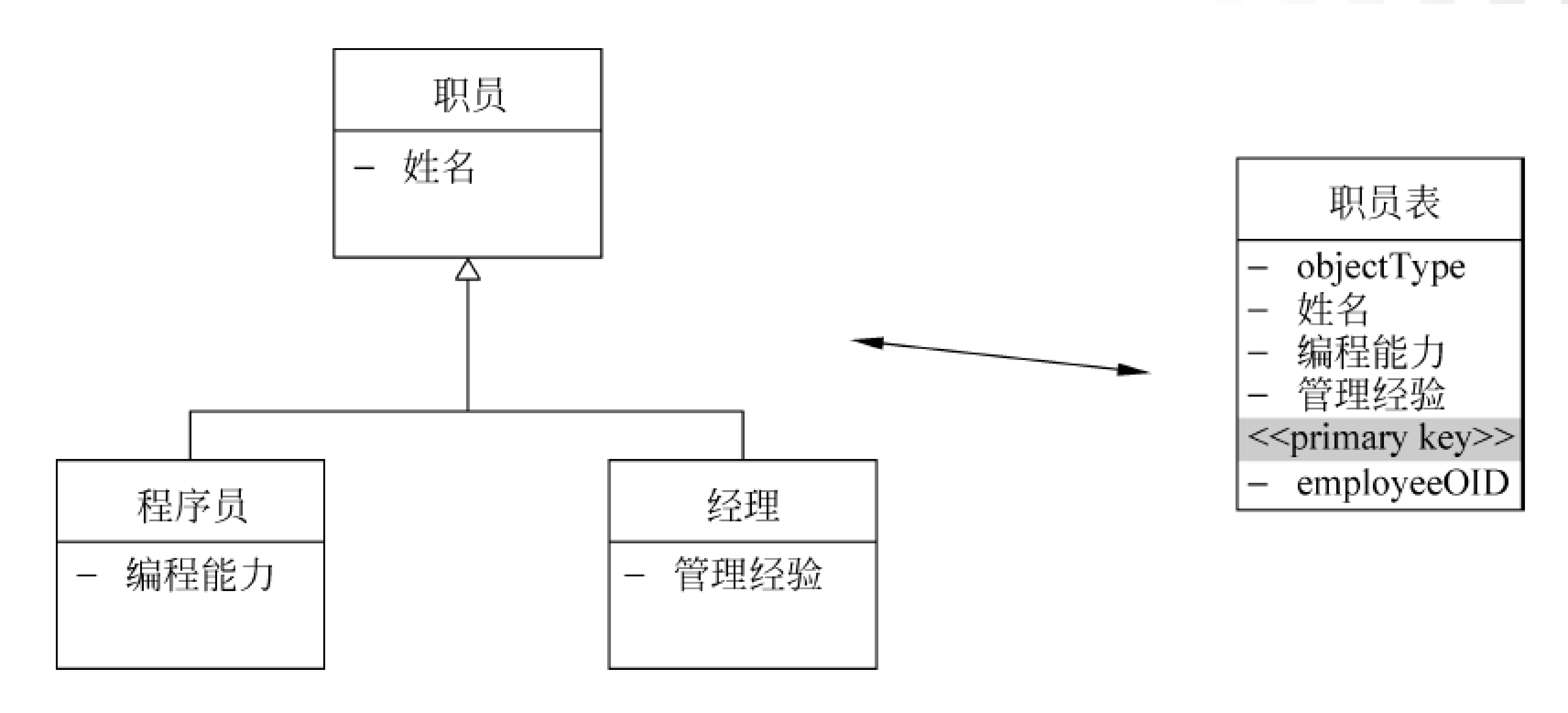


图 13-22　类层次映射到单个数据表

这种方法的优点是实现比较简单，当某一个对象角色改变时也能支持多态，需要某个职责的数据都会在同一张表中找到。其缺点也比较明显，当任何一个类添加一个属性时都需要修改表结构，如果增加的属性不正确，不仅影响新增属性子类，而且还影响到整个类层次中所有的类。这种方案也会浪费数据库中的大量空间。表中增加 objectType 字段用于区别当前行的职责角色，但如果有多重角色存在时，如有个职员既是经理又是程序员，就需要特别处理。

2. 每个具体类使用一个数据表

在上述示例中，具体类是指"程序员"和"经理"类，不包括抽象类"职员"类。在这种方法中，每个表代表着一个类，使每张数据表既包含具体类的属性，又包含继承自父类的属性，如图 13-23 所示。

这种方法访问某一类人员信息仅需访问单张表，但它也存在一些缺点。当修改抽象类的属性时，也需要修改它的任意子类的表。当一个对象改变角色，如由一个程序员变为经理时，需要把数据复制到合适的表中，并重新分配一个新的 OID。同时对于多重角色的对象维护数据完整性是比较困难的。

3. 每个类使用一个数据表

为每一个类创建一张表，父类与子类使用同一个 OID。子类表的主键同时作为父类表的外键。如图 13-24 所示，程序员表和经理表中的 employeeOID 除了是表的主键外，同时还是职员表中的外键。

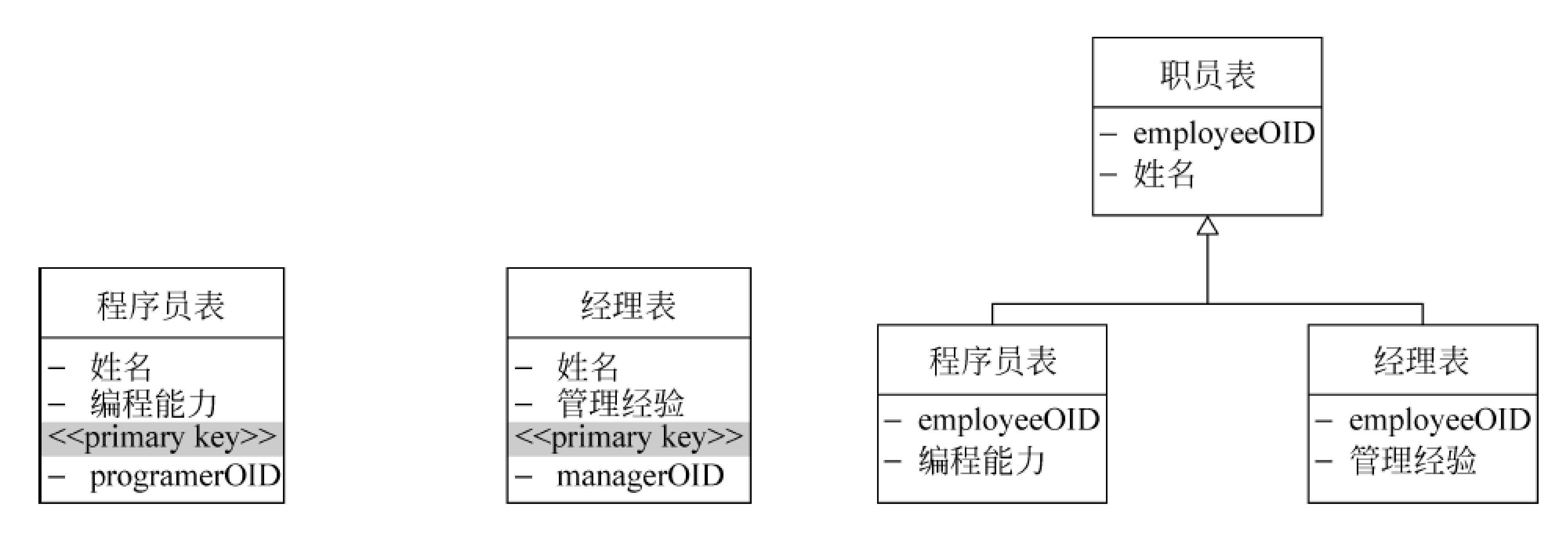


图 13-23　具体类映射单个数据表　　图 13-24　每个类映射到一张数据表

这种方案最符合面向对象的概念，当某一个类发生变化时，也很容易修改相应的表。但是，缺点是数据库里面会有许多表，读写数据的效率可能会慢一些。另外，采用这种方法还要注意灵活使用视图。

每种不同的策略产生不同的持久性模型。没有哪种策略对于所有情况都是理想的。在具体实践过程中究竟选择哪种方法，需要在具体问题上具体分析，选择最合理的解决策略。

13.6.3 关联和聚合映射

仅仅把类映射到表，属性映射到列，这样显然是不够的。不但要把类映射到数据库中，还需要映射类涉及的关系。类间的关系主要是继承、关联和聚合/组合。继承在前面已经讨论过了，聚合只是一种特殊的关联而已，从数据库角度看，关联与聚合/组合唯一的区别在于对象之间紧密绑定的程度。两种关系映射时处理方式类似，但是在读写数据操作时根据特定的业务领域会有所区别。

在关系数据库中通常使用外键实现类之间的关联。外键使一张表的一行记录与另一张表中的一行记录联系起来。映射“一对一”在关联时，需要知道关联的方向性。例如，职员与岗位之间存在“一对一”关联，职员需要知道他的岗位信息，而反过来岗位不需要知道职员的信息。因此在职员表中增加“positionOID”的外键，建立起与岗位表的关联。如果这是一个双向关联，那岗位表也就需要添加一个“employeeOID”的外键，如图 13-25 所示。

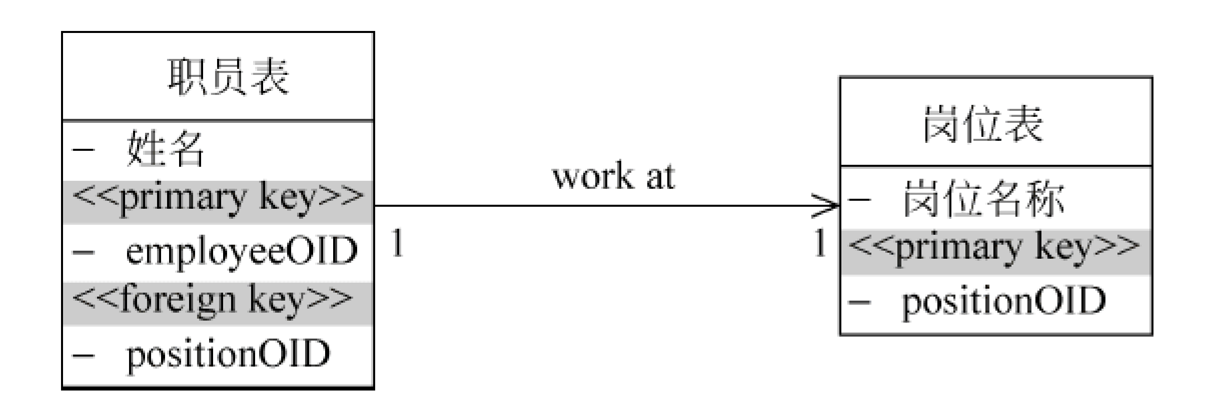


图 13-25 一对一关联映射

如果是“一对多”的关联，如“职员”类与“任务”类之间是一对多的关系，一个职员同时做多个任务，一个任务分配给一个职员完成。这时只要把职员表的主键作为外键放入任务表中，因为关系是在“多”的一方维护。

要实现多对多的关联，就需要用到关联表的概念，它的唯一目的在于维护关系数据库中两张或多张表之间的关联。关联表中包含的属性一般都是与这种关系有关的表中键的组合。在图 13-26 中，“职员”和“收益”之间存在多对多关联。引入关联表后，多对多关联转化为原有的类与关联类之间一对多的关系。

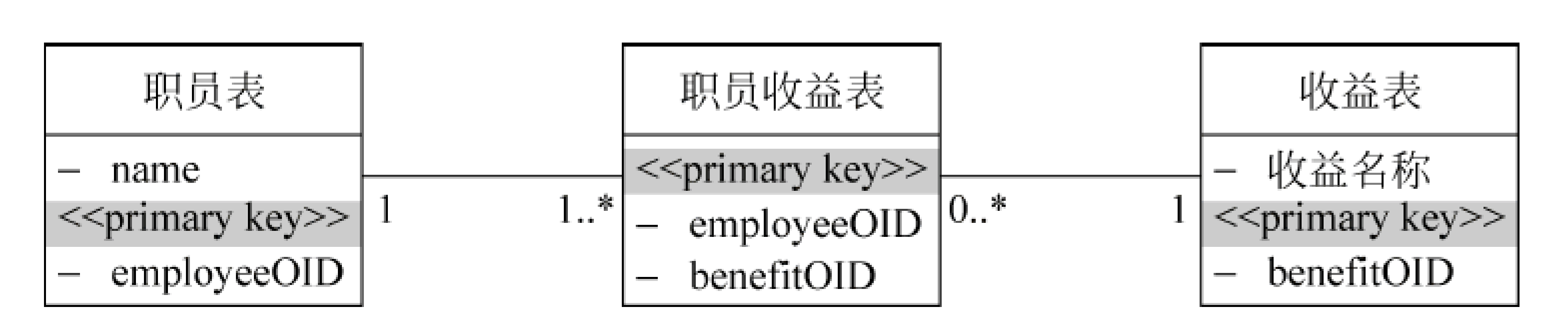


图 13-26 多对多关联映射

13.6.4 持久性框架

由于面向对象的对象表示与关系数据库的关系表示之间存在失配问题，设计一种可复用且可扩展功能的框架，提供对象与关系间的映射服务，会给编码带来很大便利。持久性框架(Persistence Framework)就是一组通用的、可复用的、可扩展的类，相互协作提供支持持久存储对象的服务，称为持久性服务，也称为对象/关系映射服务(Object/Relational Mapping，ORM)。持久性框架将应用程序与其使用和操作的数据源分离，位于数据源之上，通常将对象转换为记录，并将它们存入数据库，或从数据库中读取记录转换成对象。

目前已经存在许多稳定的、达到工业水平、免费并开源的优秀持久性框架。例如，Hibernate[①] 在 Java 领域中已被广泛使用，它几乎解决了对象—关系映射、性能、对事务的支持等关系数据库操作的所有问题。Hibernate 通过 *.hbm.xml 配置文件将关系数据库中的表映射为系统中使用到的业务持久类。对这些持久类 Hibernate 向上层提供 SessionFactory、Session、Tracsaction 等接口，可以方便地访问数据库(图 13-27)。

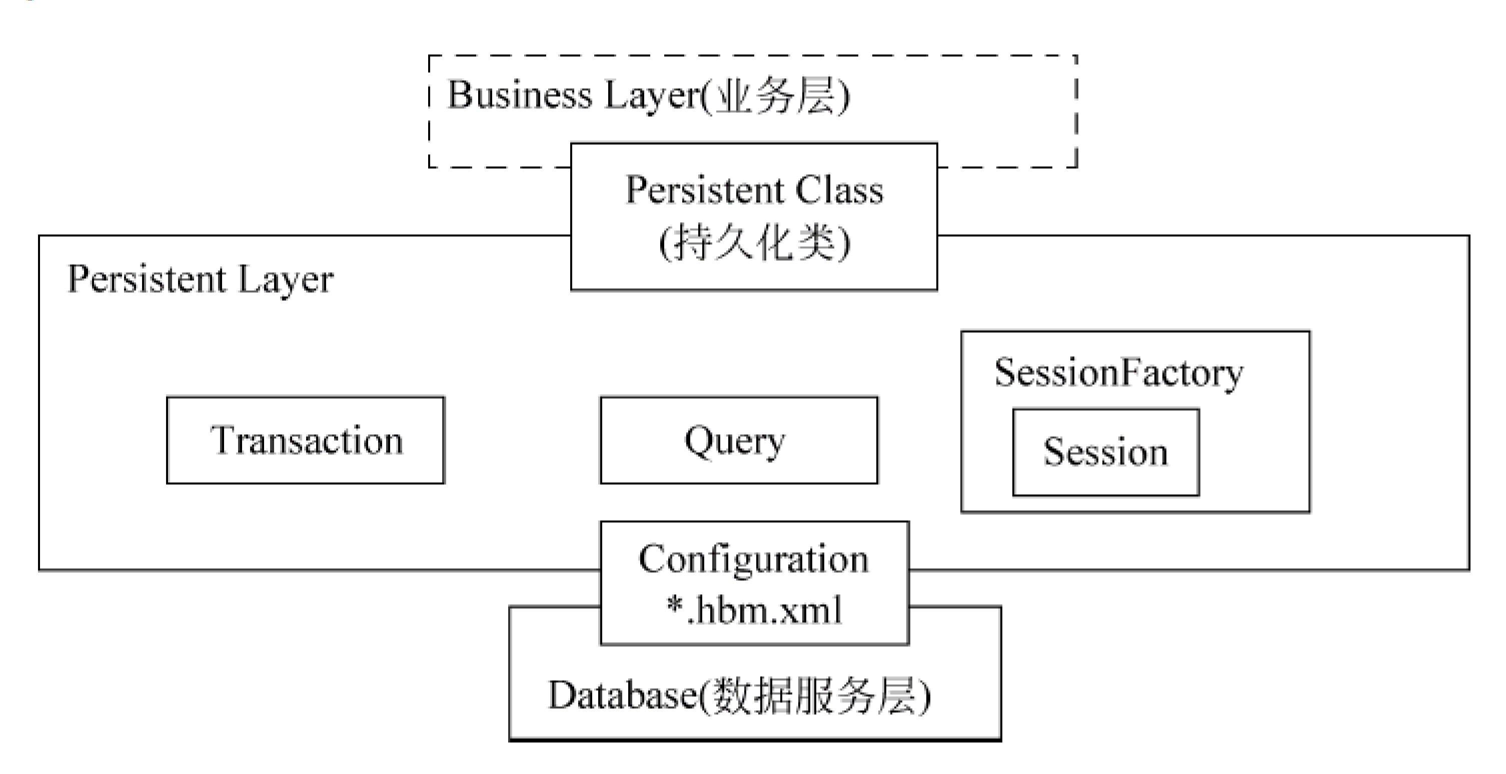


图 13-27 Hibernate 应用流程

13.7 部署建模

部署图描述了处理结点及运行在这些结点之上的构件运行时刻静态配置视图。换句话说，部署图显示了系统需要使用的硬件和在这些硬件上运行的软件，以及用于把这些硬件联系起来的中间件。应当为部署在多台机器上的系统建立部署模型，例如，销售的应用系统运行在瘦客户端上，直接访问中央数据库服务器。还有分布式对象结构，如 CORBA。嵌入式系统的设计，也同样需要部署模型，表示出硬件与软件构件如何一起工作。总之，除了很小的系统外，所有的系统都需要部署建模。

部署图中最基本的元素是结点(Node)，有以下两种类型的结点。

(1) 设备结点：具有处理和存储能力，可执行软件的物理计算资源，如典型的计算机或移动电话。

(2) 执行环境结点：在外部结点中运行的软件计算资源，其自身可以容纳和执行其他

① www.hibernate.org

可执行软件元素。如操作系统、虚拟机、信息发布系统或数据库引擎等。

分布式系统通常是由多级服务器构成的系统。部署图建模与系统采用的分布结构策略有很大关系。采用胖客户端还是瘦客户端？应用二层、三层还是更多层结构？根据分布策略确定部署的结点，以及结点间的连接方式。进一步分析软件构件应当被分布到哪个结点上。如果使用3层结构的方法将直截了当：应用程序构件在客户端，领域构件分配给应用服务器，数据库则分配数据库服务器。不同的分布策略，自然会有不同的构件分布。

嵌入式系统是软件密集的硬件集合，其硬件与物理世界连接。嵌入式系统包括控制设备（如马达、传动装置和显示器等）的软件，又包括由外部的激发（如传感器输入、运动和温度变化等）所控制的软件。嵌入式系统的部署图研究系统与应用环境里的其他系统的依赖关系。

图13-28显示出一幅3层架构的系统部署图。三维方框代表结点，结点之间的连接用简单的直线表示。

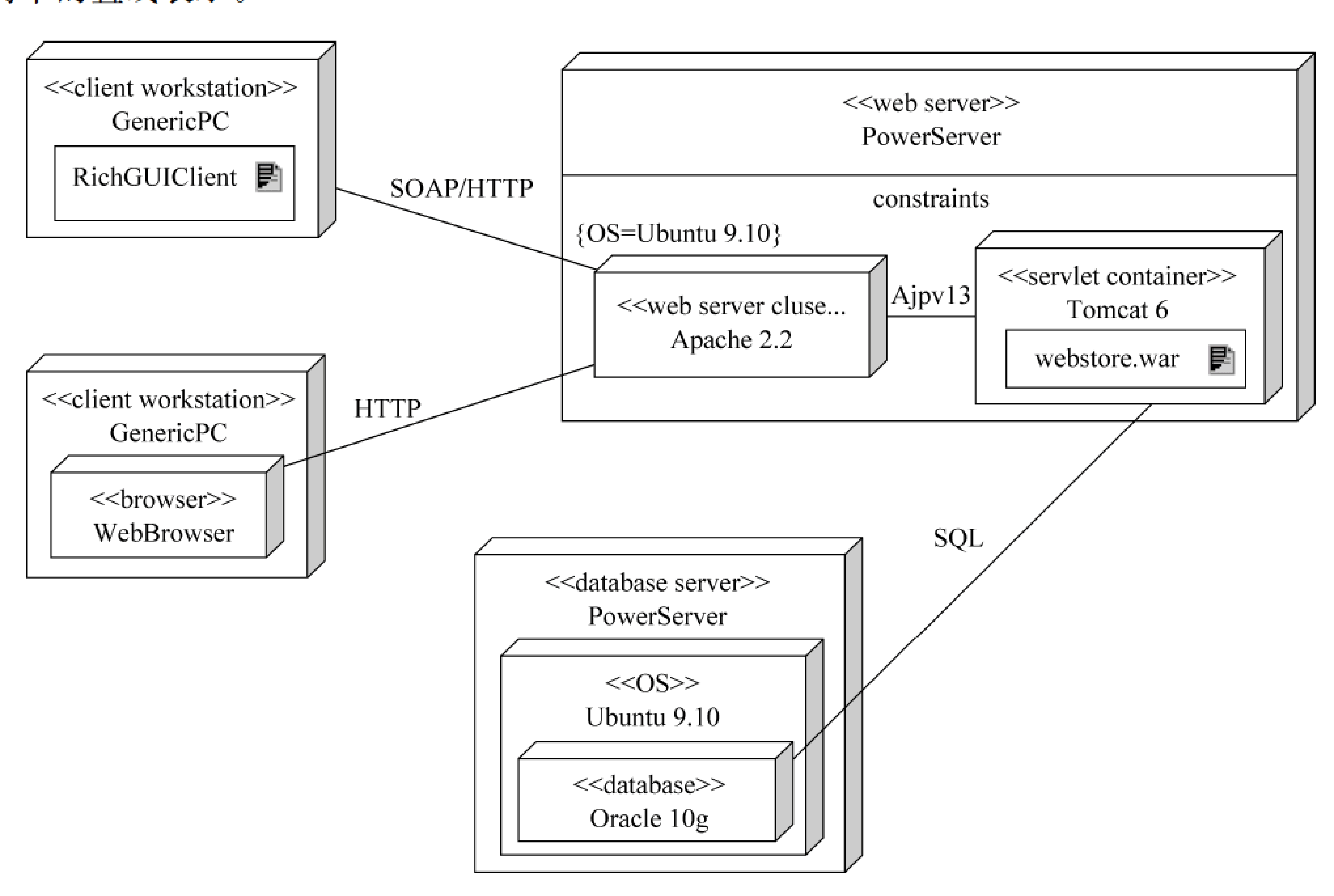


图13-28　3层架构系统的部署图

本 章 小 结

面向对象设计将分析阶段所建立的分析模型映射为可以根据特定架构编码实现的详细设计模型。OOA和OOD有着不同的侧重点和不同的分工，OOA只针对问题域和系统责任，不考虑与实现相关的因素，建立一个独立于实现的OOA模型，OOD则考虑与实现相关的问题（如选用的编程语言、软件架构、数据库和人机交互等），建立一个具体的可实现的OOD模型。

对于一个大而复杂的系统，在面向对象设计时，也要分解成若干个较小的子系统，再对

每个子系统进行分析求解。同时决定系统采用的逻辑结构，经典的3层架构即将系统分为3层，分别是数据访问层、业务逻辑层和表现层。类模型的设计详细刻画了每个类的属性和操作。在使用交互模型描述用例的实例过程时，应考虑对象间的协作过程用代码实现的每一个细节。

类模型的设计往往需要遵循一些设计原则，这些设计原则通过很小的设计改变就可以适应设计需求的变化并且能减少设计修改造成的副作用。许多面向对象专家提出很多类的设计原则，在本章中着重介绍4条设计原则：开闭原则、Liskov替换原则、依赖倒置原则和接口分离原则。同时对于类模型重构应用成熟的设计模式，使代码重用性更高，让代码更容易被他人理解，保证代码可靠性。GoF的《设计模式》提出了23种基本设计模式。本章中介绍了单件模式和抽象工厂模式。

应用程序运行时，总是有些数据必须被持久性存储，大部分的数据存储都是采用关系数据库，面向对象设计还需要考虑如何将持久对象及其关联映射成相应的关系数据模型。持久性建模往往是由类模型图驱动，系统设计还应考虑用户界面构件和系统部署模型等。

思考与练习

1. 面向对象设计与面向对象分析的区别是什么？设计包括哪些活动？

2. 考虑在类设计中，为什么建议使用setter方法和getter方法进行属性设置和读取？

3. 请从网上或图书馆查阅资料，编写小论文介绍系统体系结构除了分层方式外还有其他哪些系统体系结构？

4. 请举例说明设计原则对类设计起到什么样的帮助作用。

5. 本章中介绍了4种类的设计原则，除此之外，还有很多的类设计原则。请查阅资料，介绍你认为对面向对象设计很有帮助的设计原则，并说明理由。

6. GoF的设计模式有23种，请参考其他资料，在单件模式和抽象工厂模式之外，选择介绍一个设计模式，并举例说明如何使用该模式。

7. 考虑一个日期“Date”类，包含属性year、month和day，以及相关方法的定义。这个类的职责除了能够准确表示日期之外，还能求出与另一个日期之间相差的天数，以及能增加或减去给定数目得到新的日期。请试着给“Date”建模，对于类中每个属性和方法，讨论其可见性及方法的参数和返回值等。

8. 一个公司雇佣了若干名员工，每个员工的信息包括员工号码、姓名、地址和生日。该公司当前有几个项目，每个项目的信息包括项目名称和开始日期。每个员工可同时被分派到一个或几个项目中，也可以不做任何项目。每个项目至少由一个员工来承担。公司在每个月末给每个员工邮寄一张支票，支票上的数额与项目的性质和工作时间相关。请为上述情况建立类模型，要求详细给出类的属性、方法(参数、返回值)和类间关系(多重性、角色名)。

9. 请使用两种方法为下列情况进行持久建模：

一个教授可能会教多门课程，一门课程可以由一个或多个教授来教。

提示：一种在每个数据实体中使用外键，另一种使用关联表表示关联。比较并对照这两种方法的好处。考虑在何种情况下使用哪一种方法比较合适。

10. 在持久建模中,使用关联表分解多对一关联的优缺点是什么?

11. 一篇文章通常由许多节构成,而每节通常又由许多段落构成。

(1) 请用类图描述上述情况。

(2) 请将该类图映射为数据库中的表,并解释映射的原因。

(3) 如果考虑文章中可能包含的图和表,那么相应的类图和数据库表会有什么变化?

12. 查阅资料,了解面向对象数据库系统的产品化程度,并分析其对面向对象概念的支持程度。

13. 如图 13-29 是航空公司系统的部分类模型,模型里的类都是需要持久存储的对象,请将模型里的类映射为关系数据库里的表。

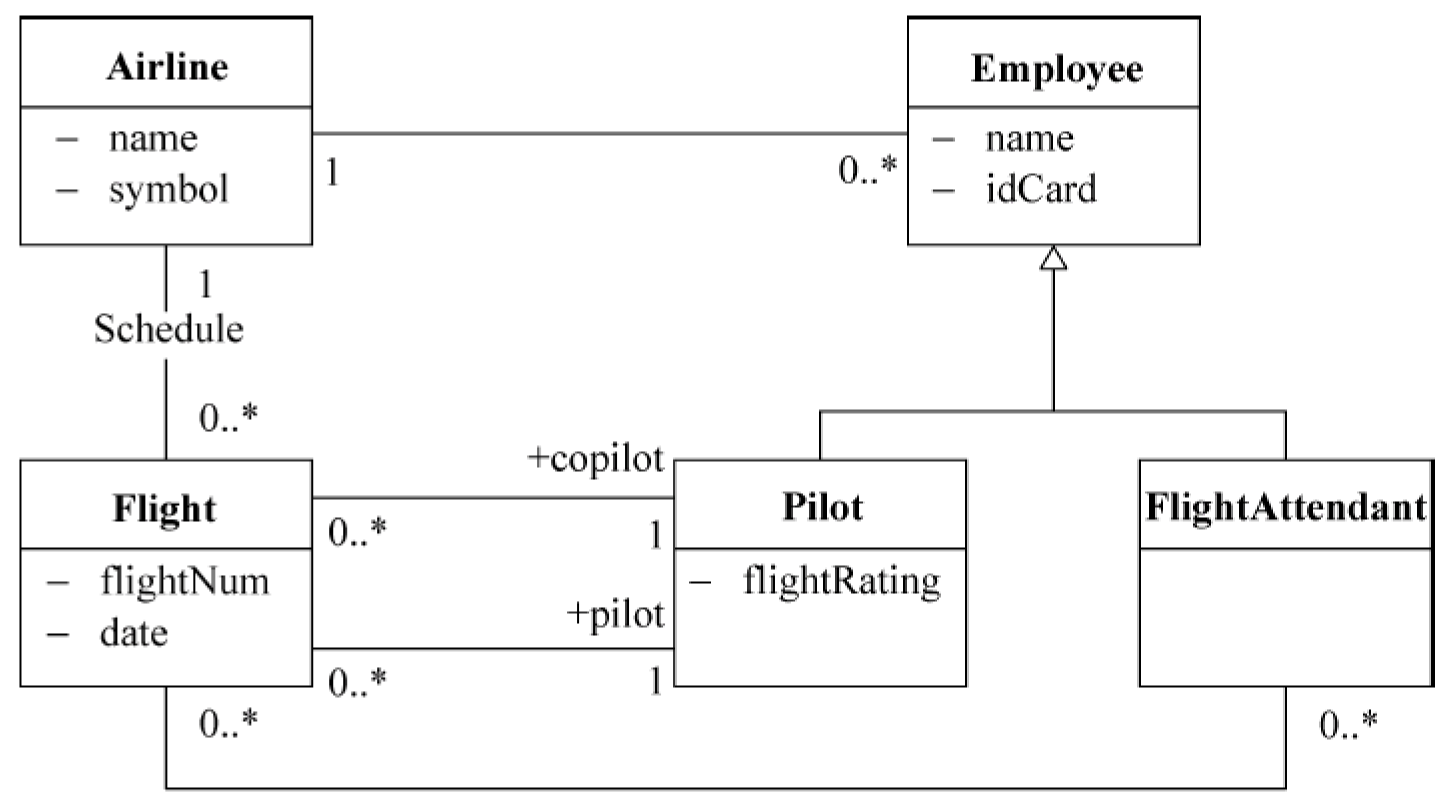


图 13-29 航空公司系统的部分类模型

14. 请给下述场景构建部署图:已知构件 AccountingComponent 是在一个 Linux 服务器上实现的,它有两个接口 UserServices 和 ManagerServices。构件 UserApps 是在一个 Windows 2000 的计算机上运行的,它通过接口 UserServices 来访问 AccountingComponent。这两台计算机运行在一个 100Mbps 的 TCP/IP 局域网上。

15. 针对你所编制过或熟知的一个系统,绘制部署图。

第14章 面向对象测试

包括模型、文档和源代码在内,任何值得构建的东西都应当测试。

——Scott W. Ambler(加拿大)

测试的目标总是在有限的时间内发现最大数量的错误。虽然面向对象测试的整体目标和传统软件测试的目标是一致的,但是测试的策略和方法有较大不同,其测试的视角扩大到了分析和设计模型的评审,测试的焦点也从过程模块转移到了类。

随着面向对象分析和面向对象设计技术的成熟,尽管面向对象软件开发技术的基本思想保证了软件应该有更高的质量,但却使系统的结构变得更加复杂。为了保证系统的高可靠性,需要更多而不是更少的测试。

面向对象程序设计语言所独有的多态,继承,封装等新特点,产生了传统语言设计所不存在的错误可能性,或者使传统软件测试中的重点不再显得突出,有可能使传统软件测试中的重点不再显得突出,也会使原来测试经验认为及经过实践证明的传统软件测试次要方面成为面向对象测试的主要问题。

面向对象软件的体系结构导致一系列分层的子系统,不再是传统的功能模块结构,它们封装了协作的类。子系统和类协作实现系统的功能。因此原有集成测试所要求的逐步将开发的模块搭建在一起进行测试的方法已不可能,有必要在各个不同的层次测试封装的系统或类,发现可能存在类相互协作或子系统跨体系结构通信时的错误。而且,面向对象软件抛弃了传统的开发模式,对每个开发阶段都有不同以往的要求和结果,已经不可能用功能细化的观点来检测面向对象分析和设计的结果。举个简单的例子:

在传统的面向过程程序中,对于函数:

```
y = Function(x);
```

只需要考虑一个函数 Function()的行为特点,而在面向对象程序中,不但要考虑基类 Function 函数的行为还要考虑子类 Function 函数的行为。

因此,传统的测试模型对面向对象软件已经不再适用。针对面向对象软件的开发特点,应该采用全生命周期面向对象测试方式。

面向对象软件的构造从分析模型和设计模型的创建开始。软件是从非形式化的系统需求表示开始,逐步增量迭代,演化为详细的类、类连接和关联、系统结构设计和职责分配等的概念模型,最终按照模型实施而成。建模基于定义阶段的需求收集,设计源于分析模型的推导,编码基于设计模型。因此,在每个阶段,模型都应该被测试,以避免错误传播到下一次迭代。

14.1 全生命周期面向对象测试

Ambler 提出的全生命周期面向对象测试(FLOOT)方法学[①]是测试技术的集合,用于验证和检验采用面向对象方法开发的软件。在图 14-1 中描述了 FLOOT 方法在软件测试所有方面提供的不同技术。从需求阶段开始,一直到软件正式使用之前,FLOOT 提倡的是针对开发过程的全方位测试,尽量在软件开发早期就进行测试,而且需要经常性的测试。

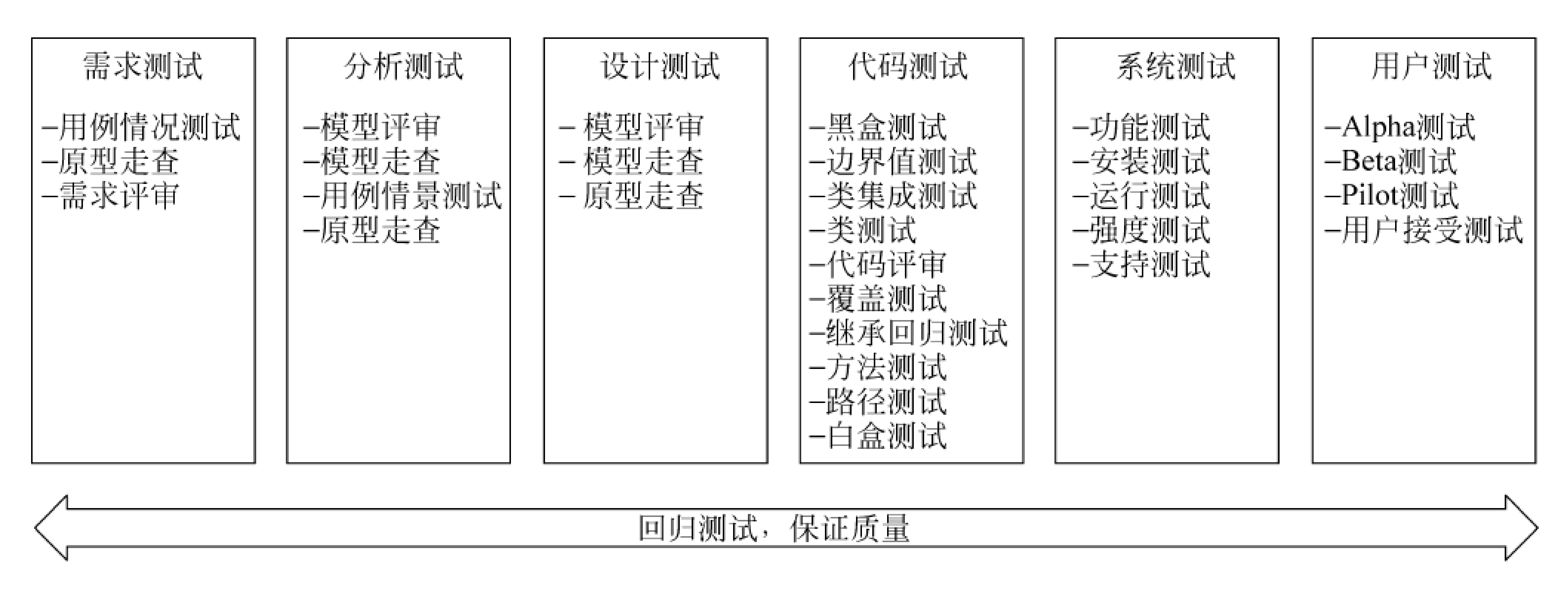


图 14-1 全生命周期面向对象测试技术

许多软件缺陷是在软件开发生命周期早期引入的,其中大部分常常是在需求和分析活动中引入的。另外,越早检测到错误,修改的成本就会越低。因此,模型测试十分重要。常使用的技术有场景测试、原型走查、用户需求评审、模型评审等。执行用例场景测试有助于验证用户需求。原型走查能很快地验证原型是否达到了用户的要求。而用户需求评审确保需求能够准确地反映用户的要求。模型评审主要组织领域专业或技术人员,评估开发出的所有分析或设计模型,以求较早地揭示开发过程中的缺陷。这些测试可以与需求、分析和设计平行进行,以期尽可能早地发现问题。

面向对象语言的特点注定了源代码测试需要新的测试技术。主要测试方法有类测试和继承回归测试等。

至于系统测试和用户测试,面向对象程序测试与传统的结构化程序测试所采用的方法差别并不大。

14.2 面向对象测试策略

传统的测试计算机软件的策略是从"小型测试"开始,逐步走向"大型测试"。换个说法就是:从单元测试开始,然后逐步进入集成测试,最后是确认测试和系统测试。在传统应用中,单元测试关注于最小的可编译程序单位——子程序(如模型、子例程、过程),一旦这些单元被逐个测试过,就可集成到程序体系结构中,然后通过一系列的回归测试检测这些新单元

① Scott W. Ambler. The Full Life Cycle Object-Oriented Testing (FLOOT) Method. http://www.ambysoft.com/essays/floot.html

加入所带来的副作用，最后将系统作为一个整体进行集成测试，以保证满足需求。

14.2.1　面向对象的单元测试

与传统单元测试相比，面向对象的单元测试中的单元的概念发生了变化。封装了属性和操作的类或对象作为最小的可测试单位，而不是原来的数据与操作分离的个体模块。类包含了一组不同的操作，并且某特殊的操作也可能作为一组不同类的一部分存在，如 ATM 系统中不同的事务操作。因此，单元测试的意义发生了较大变化。

不能按照传统的单元测试方式，孤立地测试单个操作，而是要将操作作为类的一部分考虑。考虑一个类的层次结构，其中父类定义了操作 X 并被一组子类继承。每个子类使用操作 X，但是它被应用于每个子类定义私有属性和操作的语境中。不同的语境对操作 X 的执行结果都有不同的影响，因此，有必要在每个子类的语境内测试操作 X。这意味着在真空语境中测试操作 X 在面向对象的环境下是无效的。

对于面向对象系统的类测试等价于传统软件的单元测试。传统软件的单元测试往往关注模块的算法细节和模块接口间数据的流动。面向对象的类测试是由封装在类中的操作和类的状态行为驱动的。

14.2.2　面向对象的集成测试

由于面向对象系统没有层次的控制结构，传统的自顶向下和自底向上的集成策略就没有了意义。此外，由于类的直接或间接交互构成系统，系统集成不可能像传统的增量集成，将操作逐个集成到类中。

对于面向对象系统的集成测试有两种不同策略：第一种称为基于线程的测试（Thread-Based Testing），集成响应系统的一个输入事件所需要的一组类，每个线程被逐个集成和测试，用回归测试以保证集成没有产生副作用；第二种集成方法称为基于应用的测试（Use-Based Testing），通过测试那些几乎不需要其他类协作的类（称为独立类）开始系统的构建，在独立类被测试后，测试第一层依赖于独立类的类（称为依赖类）。随着依赖类的测试层次序列继承增加，最终构建出完整的系统。与传统的集成测试不同，面向对象集成测试不需要编写额外的辅助程序协助测试。

集群测试（Cluster Testing）是面向对象系统集成测试的一步，是一组相互协作的类群通过设计模型发现协作中的错误而采取的测试方式。

14.2.3　面向对象的确认测试

在确认测试时，无需关心系统内部的层次以及类之间的连接细节。与传统确认测试一样，面向对象软件的确认测试关注用户可见的动作和用户可识别的系统输出。测试员可以利用分析模型的一部分，如用例图提供的场景，确认系统行为。

传统的黑盒测试方法可被用于驱动确认测试，此外，测试用例可以从创建面向对象分析模型的一部分，如用例场景和交互模型等导出。

14.3 基于场景的模型测试

分析与设计模型只是概念上的模型，不能像程序一样执行，所以不能进行传统意义上的测试。大部分采用技术评审的方式检查分析与设计模型的正确性和一致性。通常采用用例场景的方式测试分析和设计模型。

场景测试是一种非常有效的测试模型正确性的手段之一。提出这种测试思想的是Rational 公司，最早在 RUP 2000 中文版当中有其详尽的解释和应用。基本的思想是使用一系列用例场景，巡查分析或设计模型，按照场景演练模型，验证这些模型能否支持这些场景。如果不支持，就要适当地修改模型。

场景指的是用户使用系统某项功能时与系统交互的操作流程，是通过描述流经用例的路径来确定的过程，这个流经过程要从用例开始到结束遍历其中所有基本流和备选流。

现在的软件几乎都是由事件触发来控制流程的，事件触发时的情景便形成了场景，而同一事件不同的触发顺序和处理结果形成事件流。这种在软件设计方面的思想也可以被引入软件测试中，生动地描绘出事件触发时的情景，有利于测试设计者设计测试用例，同时测试用例也更容易得到理解和执行。

场景测试除了用于测试模型以外，生成的测试数据也可对集成后的系统进行验证测试。

14.3.1 场景设计

虽然场景在概念上与用例相似，但实际上是有差别的。对于用户来说，用例是一种有意义的单一内聚任务的描述，而场景描述的用户使用系统某一场景，可以是一个用例基本活动过程，也可能使用场景跨越了多个用例。如用户使用 ATM 系统实现转账功能的用例场景，实际上跨越了“验证身份”、“转账”和“打印回执”3 个用例。

如果是单个用例，只需根据用例规约的描述分析场景。当场景跨越多个用例时，场景设计与单个用例类似，先定义出每个用例的使用场景，然后再将每个用例的场景串接在一起。

图 14-2 中经过用例的每条不同路径都反映了基本流和备选流，使用箭头来表示。基本流用直黑线来表示，是经过用例的最简单的路径。每个备选流自基本流开始，之后，备选流会在某个特定条件下执行。备选流可能会重新加入基本流中(备选流 1 和 3)，还可能起源于另一个备选流(备选流 2)，或者终止用例而不再重新加入某个流(备选流 2 和 4)。

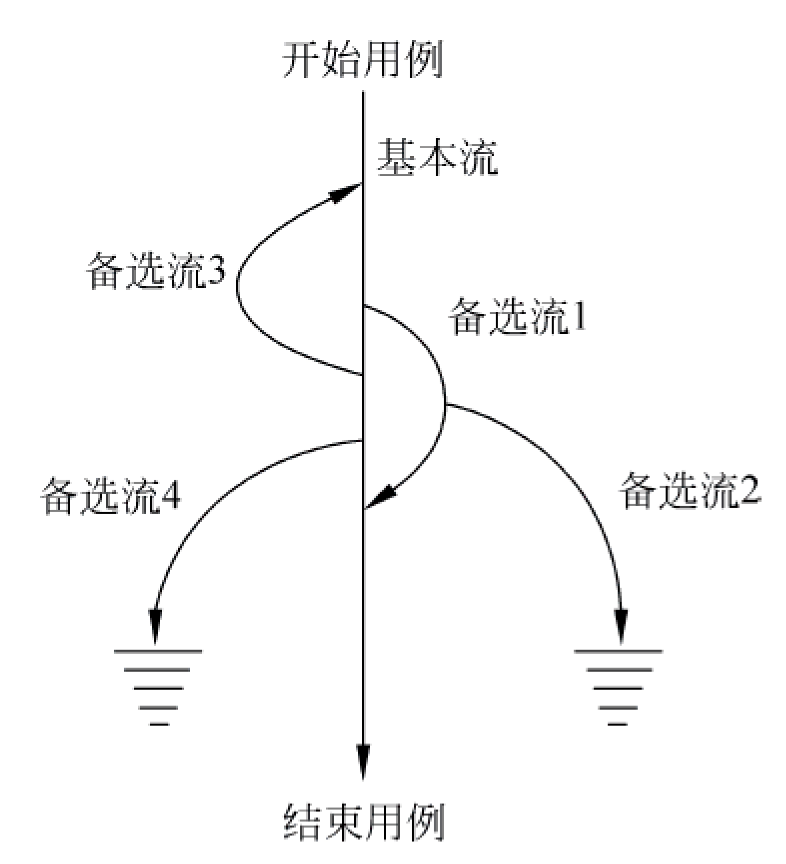


图 14-2 基于场景的模型测试技术

遵循图 14-2 中每个经过用例的可能路径，可以确定不同的用例场景。从基本流开始，再将基本流和备选流结合起来，可以确定以下用例场景：

- 场景 1：基本流。
- 场景 2：基本流，备选流 1。
- 场景 3：基本流，备选流 1，备选流 2。
- 场景 4：基本流，备选流 3。

- 场景 5：基本流，备选流 3，备选流 1。
- 场景 6：基本流，备选流 3，备选流 1，备选流 2。
- 场景 7：基本流，备选流 4。
- 场景 8：基本流，备选流 3，备选流 4。

注：为方便起见，场景 5、6 和 8 只描述了备选流 3 指示的循环执行一次的情况。

生成每个场景的测试用例是通过确定某个特定条件完成的，这个特定条件将导致特定用例场景的执行。

在第 12 章中给出了转账用例的用例规约，规约中也给出了成功场景和失败场景，按照上述方法，归纳出转账用例的场景，如表 14-1 所示。

表 14-1 转账用例的所有场景

场景号	场 景 名	场景执行流程
场景 1	成功转账	基本流
场景 2	取消转账	基本流中的某个步骤客户选择了“取消”操作
场景 3	操作输入超时	基本流中的某个步骤客户输入超时
场景 4	无效转账账号	基本流第 5 步骤中，客户输入无效转账账号
场景 5	转账金额超过限额	基本流第 6 步骤中，客户输入转账金额超限
场景 6	转账金额超过卡内余额	基本流第 6 步骤中，客户输入转账金额超出卡内余额

一旦用例场景有了明确的定义，需要验证系统能否按照场景的描述，帮助用户达到使用目的。对于需求、分析和设计的模型，使用场景都可以验证其正确性。

14.3.2 测试用例设计

根据每个场景，设计确认测试的用例。以 ATM 的“转账”用例为例，表 14-1 所示的 6 个场景中的每一个场景都需要确定测试用例。可以采用矩阵或决策表来确定和管理测试用例。表 14-2 显示了一种通用格式，表中一行代表每个测试用例，而每列则代表测试用例使用到的信息。在本示例中，对于每个测试用例，存在一个测试用例 ID、场景说明、测试用例中涉及的所有数据元素（作为输入或已经存在于数据库中）以及预期结果。

表 14-2 转账测试用例表

测试用例 ID	场 景 说 明	转账账号	转账金额	账户余额	账户转账限额	预 期 结 果
TC1	成功转账	√	√	√	√	转账成功
TC2	取消转账	N/A	N/A	N/A	N/A	提示取消信息，用例结束
TC3	操作输入超时	N/A	N/A	N/A	N/A	提示超时信息，用例结束
TC4	无效转账账号	I	√	√	√	提示账号无效，返回基本流步骤 5
TC5	转账金额超过限额	√	I	√	√	提示金额超限，返回基本流步骤 6
TC6	转账金额超过卡内余额	√	I	√	√	提示金额超过余额，返回基本流步骤 6

表中"√"符号代表条件符合要求,"N/A"代表条件不适用,"I"代表数据是可修改的输入项。

一旦确定了所有的测试用例,则应对这些用例进行复审和验证以确保其准确且适度,并取消多余或等效的测试用例。测试用例一经认可,就可以确定实际数据值(在测试用例实施矩阵中)并且设定测试数据,如表 14-3 所示。

表 14-3　转账测试用例数据表

测试用例 ID	场 景 说 明	转账账号	转账金额	账户余额	账户转账限额	预 期 结 果
TC1	成功转账	×××××	1000	8000	20000	转账成功
TC2	取消转账	N/A	N/A	N/A	N/A	提示取消信息,用例结束
TC3	操作输入超时	N/A	N/A	N/A	N/A	提示超时信息,用例结束
TC4	无效转账账号	×1	1000	8000	5000	提示账号无效,返回基本流步骤 5
TC5	转账金额超过限额	×××××	6000	8000	5000	提示金额超限,返回基本流步骤 6
TC6	转账金额超过卡内余额	×××××	3000	2000	5000	提示金额超过余额,返回基本流步骤 6

14.4　类测试

面向对象程序由类组成,每个类包括方法(操作)、属性以及与其他类的联系。类的测试是检查类的代码是否完全满足类说明所描述的要求。如果类的实现正确,那么类的每一个实例的行为也应该是正确的。对于类的测试既是单元测试也是传统的集成测试。因为类和它的实例作为单元隔离测试,但有时需要验证类的方法和属性是否可以配合工作。

14.4.1　类的代码检查

代码检查也称为代码评审,经常会暴露出普通测试技术没查出的一些问题,尤其是一些拙劣的编码实践将会使应用程序难以扩展和维护。代码检查检验是否正确地构建了代码,并检验构建的代码是否易于理解、维护和完善。因为检查会暴露出需要改善的地方,所以代码检查应尽可能在编码过程的早期进行。先写出一千行代码,然后检查、修改。再写十万行代码,然后找出对其他人来说不易理解的代码,修改完善。逐次检查,直到系统完成。

代码检查应该关注以下质量问题。

(1) 代码能满足设计吗?

(2) 类、方法和属性的命名规范性。

(3) 代码说明文档标准和规范化,代码中注释清晰。

(4) 类的方法仅实现一项功能,保证功能内聚。

(5) 代码有办法再简化吗?

14.4.2　覆盖和路径测试

覆盖测试的目的是设计出测试所有代码路径的测试用例。这些用例至少检查程序中每行代码一次。路径测试是覆盖测试的子集,不仅确保所有的代码行都测试过,而且所有的逻辑路径也都被测试过。覆盖测试确保所有的代码行都被测试,路径测试确保所有逻辑路径都被测试。

覆盖测试最主要的优点在于有助于确保应用程序中所有代码都被测试过,但它不能保证所有的代码的组合都被测试过。另一方面,路径测试确实也测试了代码组合,但它需要更多的努力来确定并运行测试用例。

14.4.3　类的随机测试

类的随机测试主要根据对象的生命周期状态转变,设计出类方法的测试序列,从而充分验证对象在整个生命周期中所有操作的正确性。如某银行应用系统中 Account 类如图 14-3 所示,有下列操作:open(开户),activate(激活),deposit(存款),withdraw(取款),balance(查询余额),suspend(挂失),reuse(取消挂失),close(销户)和 clear(清除)。这些操作在执行时隐含着某些限制,例如,账号必须开户被激活后再可以进行金融交易,顾客不需要账户时要关闭账户。从 Account 类的状态图分析,Account 实例存在的最小行为生命周期包含以下操作:

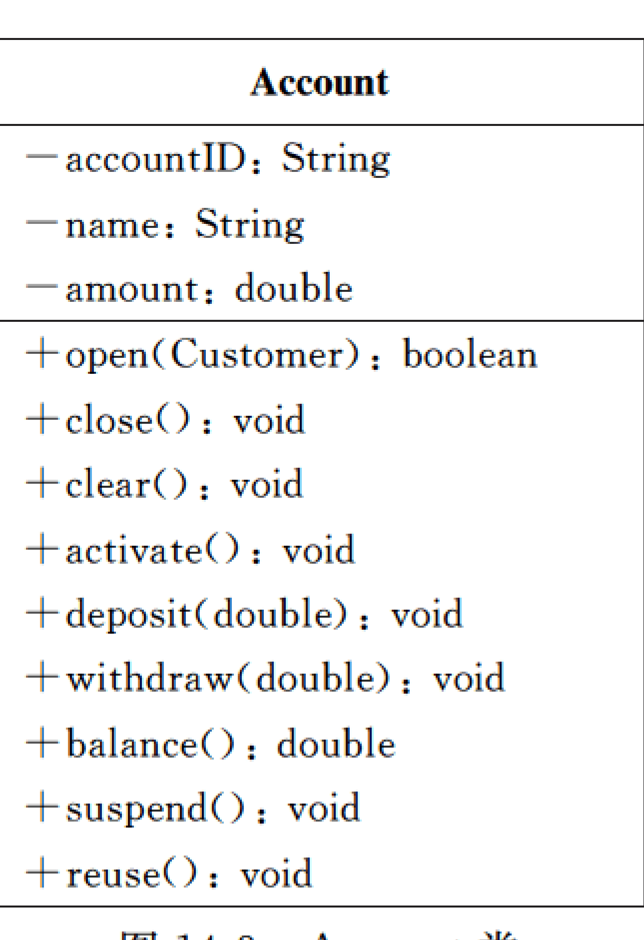


图 14-3　Account 类

open · close · clear

这表示了 account 类的最小测试序列,然而,其他行为可以加入到这个序列中:

open · [activate · [withdraw | balance | deposit | [suspend | reuse]^n]^n | suspend] · close · clear

因此,可以从上面的操作序列中,随机产生一个类的测试序列作为类的测试用例,例如:

测试用例 1:open · activate · deposit · withdraw · close · clear

测试用例 2:open · activate · deposit · withdraw · balance · deposit · balance · suspend · reuse · balance · deposit · suspend · close · clear

这些测试用例还有其他的随机测试用例可以测试不同的类实例的生命周期。

14.4.4　类的划分测试

划分测试是为了减少测试类所需的测试用例的数量,采用的是基本与传统软件的等价划分方式。将类的输入和输出分类,设计的测试用例分别测试每个类别。这种类的划分测试有三种划分方式。

第一种是基于状态的划分。这是基于类操作改变类的状态的能力来划分类的操作。例如 Account 类,改变其状态的有 activate、suspend、reuse 和 close,而非状态操作包括 balance、withdraw 和 deposit。测试用例设计分为两类,一类独立测试改变类状态的操作和另一类不改变类状态的操作:

测试用例 c1：open • activate • suspend • reuse • close • clear

测试用例 c2：open • activate • deposit • withdraw • balance • close • clear

测试用例 c1 都是改变状态的操作，而测试用例 c2 除了最小测试序列中的操作外，都是不改变状态的操作。

第二种是基于属性的划分。基于操作使用的属性对类操作分类。对 Account 类来说，属性 amount 存放账户余额信息。操作可以分为 3 个类别：①使用 amount 属性的操作；②修改 amount 的操作；③不使用或不修改 amount 属性的操作。然后对每个划分设计测试序列。

第三种是基于类别的划分。基于操作类属划分，例如，在 Account 类中的操作可被分类为初始化(open、activate)、计算操作(deposit、withdraw)、查询操作(balance)、异常处理操作(suspend、reuse)、终止操作(close、clear)。

14.5 类间测试

当面向对象的系统开始集成时，需要设计类间协作的测试用例，测试用例的设计变得更复杂。与单个类的测试用例设计一样，类协作测试用例也可以通过随机和划分方法来设计，还有基于场景的测试和行为测试。

14.5.1 多个类测试

可以按照下列的步骤生成多个类的随机测试用例。

(1) 对每个客户类，使用类操作列表生成一系列随机测试序列，这些操作将发送消息给其他服务类。

(2) 对生成的每个消息，确定服务类以及相应的服务操作。

(3) 确定被客户类的消息调用的服务对象中的操作所发送出的消息。

(4) 对每个消息，确定下一层被调用的操作并将这些操作添加到测试序列中。

在图 14-4 中，描述了 ATM 与银行后台服务系统的协作，图中的箭头方向指明了消息的方向，标注则指明隐含了调用的操作。考虑 BankServer 类为 ATM 类提供服务的操作序列：

verifyAccount $\cdot$ [withdrawReq | balanceReq | transferReq]^n

所以，对于 BankServer 类的随机测试用例可能是：

测试用例 c1：verifyAccount • withdrawReq • balanceReq

在设计测试用例 c1 时，也要考虑涉及该测试的协作者，BankServer 类必须和 BankAccount 类协作才能执行 verifyAccount、withdrawReq 和 balanceReq。因此，重新设计测试用例 c1 后，操作序列为：

测试用例 c2：$\text{verifyAccount}_{\text{BankServer}}$ • $\text{validAccount}_{\text{BankAccount}}$ • $\text{withdrawReq}_{\text{BankServer}}$ • $\text{withdraw}_{\text{BankAccount}}$ • $\text{balanceReq}_{\text{BankServer}}$ • $\text{balance}_{\text{BankAccount}}$

多个类划分测试用例设计方法类似于单个类划分测试方法，只是要将那些发送给协作类的消息而被激活的操作，添加到操作序列中。另一种方法基于不同客户类的接口来划分测试用例。例如，BankServer 类既接收来自 ATM 类也接收来自 Bank 类的消息，因此，BankServer 类中的操作可以将它们划分为服务于 ATM 类和服务于 Bank 类，而后再进行扩展。

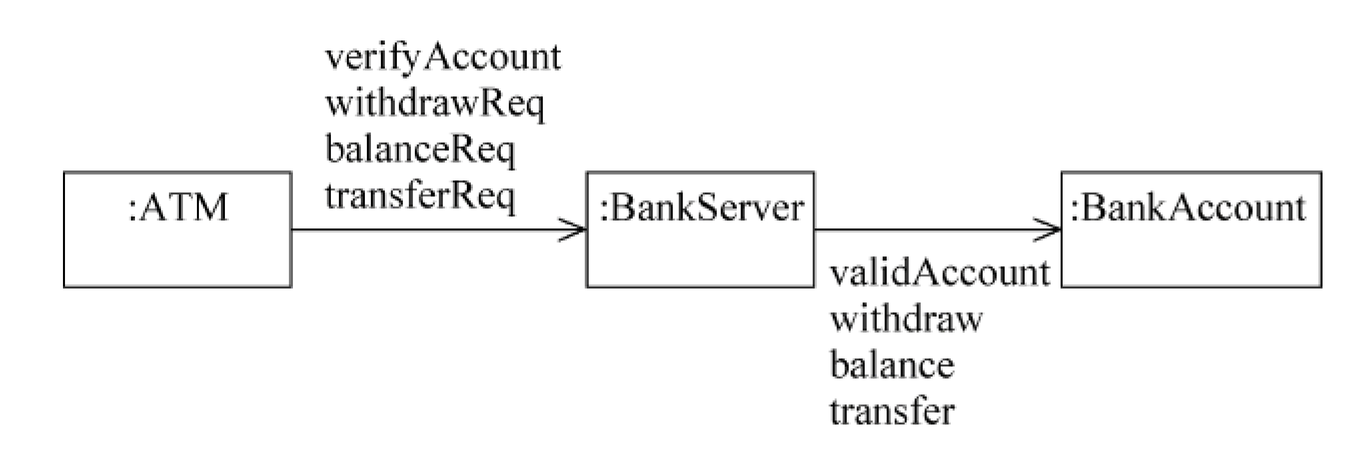


图 14-4 ATM 与银行服务系统的协作图

14.5.2 从类的行为模型导出的测试

状态图往往为类的动态行为建模。类的状态图也有助于导出测试类的动态行为的测试序列。图 14-5 给出了 Account 类的状态图。根据图上描述,类主要的状态有 4 个:“未激活”、“正常”、“挂失”和“失效”。类实例的大多数行为发生成“正常”状态,“close”操作使 Account 类进入“失效”状态,直至被清除。

从状态图出发设计的测试用例应该达到完全的状态覆盖,即操作序列应该导致 Account 类的变迁穿越所有允许的状态。

测试用例 c1:open • activate • suspend • close • clear

这个序列是穿越所有 Account 类的状态的最小序列,加入其他的操作就变成其他的测试用例。

测试用例 c2:open • activate • deposit • deposit • balance • suspend • close • clear

测试用例 c3:open • activate • deposit • withdraw • suspend • close • clear

从最小序列中可以导出更多的测试用例,用于保证类的所有行为都被充分测试过。在类的行为导致与一个或多个类协作的情况下,可以根据多个类的状态图跟踪系统的行为流程。

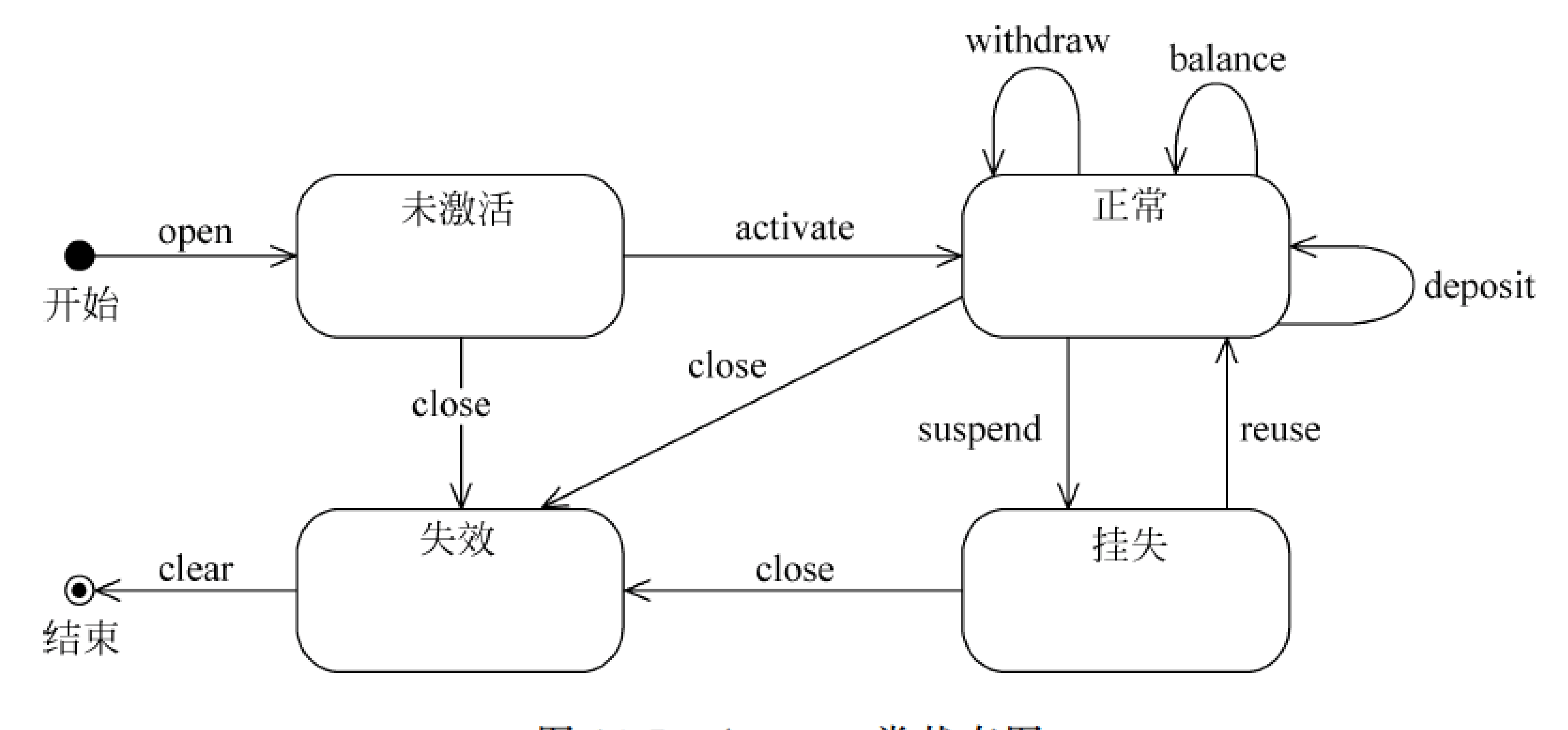


图 14-5 Account 类状态图

本 章 小 结

面向对象测试的整体目标和传统软件测试的目标是一致的。但是由于面向对象系统的构建特点,面向对象测试的策略和方法有较大不同。测试的视角扩大至软件开发整个生命周期,此外,测试的焦点从模块转向了类。

面向对象软件的构造从分析模型和设计模型的创建开始。软件是从非形式化的系统需求表示开始，逐步增量迭代，演化为详细的类、类连接和关联、系统结构设计和职责分配等的概念模型，最终按照模型实施而成。建模基于定义阶段的需求收集，设计源于分析模型的推导，编码基于设计模型。因此，在每个阶段，模型都应该被测试，以避免错误传播到下一次迭代。Ambler 提出的全生命周期面向对象测试(FLOOT)方法提倡的是针对开发过程的全方位测试、尽量在软件开发早期就进行测试，而且需要经常性的测试。

基于场景的模型测试是一种非常有效测试面向对象分析和设计模型正确性的手段之一。对于需求、分析和设计的模型，使用场景都可以验证其正确性。

一旦编码实现了面向对象软件系统，需要对每个类进行单元测试。类的测试是检查类的代码是否完全满足类说明所描述的要求。如果类的实现正确，那么类的每一个实例的行为也应该是正确的。对于类的测试既是单元测试也是传统的集成测试。因为类和它的实例作为单元隔离测试，但有时需要验证类的方法和属性是否可以配合工作。类测试使用一系列不同的方法：随机测试和划分测试等。

面向对象的集成测试是针对多个类的协作进行的。多个类的协作测试用例也可以通过随机和划分方法来设计，还有基于场景的测试和行为测试。

思考与练习

1. 试说明面向对象的软件测试与传统的软件测试有何不同。

2. 请说明为什么类是面向对象系统中测试的最小单位而不是类里的方法。

3. 为什么测试应该从面向对象分析和设计的活动开始，而不是只测试实现的代码。

4. 比较黑盒测试与白盒测试，结合方法测试、类测试以及类集成测试给出如何使用这两项技术的例子。

5. 为下面的“ShoppingBasket”类定义一组随机测试用例和划分测试用例，并比较这两种不同测试用例设计方法生成的测试用例的差别。

ShoppingBasket
－shoppingBasketNumber：String
＋addLineItem() ：void ＋createNewBasket() ：void ＋deleteItem() ：void ＋processOrder() ：void < property get > ＋getLineItem() ：LineItem < property set > ＋setLineItem(LineItem) ：void

6. 请仿照本章场景测试用例设计过程，为 ATM 系统的“密码修改”用例设计场景测试用例，并给出测试数据表。

第15章 面向对象系统的技术度量

当你能够度量所说，并将其用数字表达出来，你就能对之有一些了解；但当你不能度量，不能用数字表达时，你的了解就会处于贫乏而不满意的状态；它可能是知识的开始，但你在思想上还远没有进入科学的阶段。

——Lord Kelvin（英国著名物理学家、发明家）

在前面的章节中提到测度和度量是任何工程学科的关键构成成分，面向对象软件工程也不例外，面向对象度量是 OO 技术不可分割的一部分，在 OO 软件开发中具有重要的作用。利用度量能定量理解系统的体系结构和详细设计，利用度量的反馈信息以构造质量更好的系统；提供 OO 项目开发成本估算和进度预计的良好基础；有助于确定各种软件开发策略的作用和效果。但是对面向对象（OO）系统的度量却比其他 OO 方法使用的进展要慢得多。随着 OO 系统变得更普遍深入，软件工程师具有量化的机制来评估体系结构设计和构件级设计的质量变得更为重要，这些测度使软件工程师能在过程的早期评估软件，进行修改以减少复杂性和改善终端产品的长期生存能力。

根据面向对象度量的特点，在满足一般度量理论的基础上，构造面向对象程序所适用的度量还应该遵循以下几个准则。

（1）度量需要反映某一个面向对象的特征（继承、封装、多态）。

（2）对于面向对象程序每一个方面的特征，要从单个类和整个面向对象系统两个角度进行度量。

（3）单一的度量可能不能全面描述某一个面的特征，必须用一组度量从多个方面来反映。

15.1 OO 度量的识别特征

任何产品的度量都取决于产品的特性。OO 软件与使用传统方法开发的软件有着本质不同，因此，对 OO 系统的技术度量必须调整到那些区别 OO 和传统软件的特征上。Berard 定义了 OO 软件的 5 个特征[①]。

---

① Berard E.. Metrics for Object-Oriented Software Engineering, an Internet posting on comp. Software-eng, January 28, 1995

15.1.1 局部化

局部化是软件的一个特征，指明了信息被集中在一个程序内的方式。

(1) 传统方法中的过程模块，其数据与过程分离、功能分解，实现了功能局部化。传统软件工程使用数据驱动功能，其度量着重放在模块内部结构或复杂性(如模块规模、内聚性、环复杂度等)或模块间相互连接的方式(如模块耦合)上。

(2) 在 OO 系统中，信息是通过将数据和处理封装在类或对象边界内被集中的。类是 OO 系统的基本单位，对象封装数据和过程，所以局部化是基于对象的，因此，要把类(对象)作为一个完整实体进行度量。而且，操作和类之间的关联不一定是"一对一"的，所以，类合作的度量必须能适应"一对多"或"多对多"的关联。

15.1.2 封装

Berard 将封装定义为"一组项的包装"。

(1) 传统软件中的记录、数组只有数据没有过程，是低层次的封装；子程序(如过程、函数、子例程、段等)只有过程没有数据，是中层的封装。其度量的重点在代码行的计数和环复杂度。

(2) 在 OO 系统中，封装包含了类的职责(包括类的属性和操作)以及特定的类属性值定义的类的状态。其度量的重点不是单一的模块，而是包含数据和过程的包，是在较高抽象层次上的度量。

15.1.3 信息隐蔽

信息隐蔽是指隐藏了程序构建的操作细节，只将访问该构件所必需的信息提供给那些访问该构件的其他构件。

在这个属性上，OO 方法和传统方法是一致的。

15.1.4 继承

继承是指一个对象的属性和操作能够传递给其他对象的机制。继承发生在类层次的所有层面上。通常，传统软件不支持这种特性。继承是 OO 系统的一个重要的关键特性，因此，很多 OO 系统的度量以此为重点，包括子女的数量(类的直接子类的数量)、父辈数(直接上层的数量)、类的嵌套层次(在一个继承层次中类的深度)等。

15.1.5 抽象

抽象是使设计者只关心程序构件的主要细节(数据和过程)，而不考虑底层细节的一种机制。抽象在 OO 和传统开发中都能用到。

抽象是一种相对概念，处于抽象的较高层次时，可以忽略更多的细节，只提供一个关于概念或项的更一般的视图；处于抽象的较低层次时，可以引入更多的细节，即提供一个关于概念或项的更详细的视图。

在 OO 系统中，类从许多不同的细节层次上并以许多不同方式(如作为一个操作列表、

一个状态序列或是一系列协作)来观察,因此,OO 度量可以用一个类度量的项作为抽象的表示,如每个应用类实例化的数量,每个应用类被参数化的数量,以及类被参数化的数量与未被参数化的数量的比例等。

15.2　分析、设计模型的度量

在第 11 章中介绍过,OOD 是以 OOA 模型为基础,两者之间不存在转换,只需做必需的精化和调整,进一步设计某些细节,并增加与实现相关的部分即可。因此 OOA 和 OOD 实际上没有存在很明显的界限,两个阶段是紧密衔接的。这里度量也不再分为对分析模型和对设计模型的度量。

在关于 OO 系统的软件度量的讨论中,Whitmire 提出了以下关于 OO 设计的独特的且可测度的 9 个特征。

(1) 规模。规模可以从总数量、容量、长度和功能上来定义。总数量通过对 OO 实体(类或操作)的静态计数来度量;容量度量和总数量相同,但它是在给定的时间点动态收集;长度是对互连的设计元素链的度量(如继承树的深度);功能提供了一个特定的 OO 应用系统交付给用户的价值的间接表示。

(2) 复杂性。与规模一样,存在着多种不同复杂性观点。Whitmire 通过 OO 设计中的类的关联,从结构特性上对复杂性进行度量。

(3) 耦合度。OO 设计中元素之间的物理连接,如类间的合作数量或对象间传递的消息的数量都表示了一个 OO 系统内的耦合度。

(4) 充足性。Whitmire 将充足性定义为“从当前应用的观点看,一个抽象拥有其所需特征的程度,或一个设计构件拥有其抽象特征的程度”,简单地说,就是“该抽象(类)为了对我有用,需要拥有哪些特征”。本质上,一个设计构件是充足的,只要它完全反映了它所建模的应用领域对象的所有特征,即该抽象拥有它所需要的特征。

(5) 完备性。完备性与充足性的唯一不同在于“用于比较的都是抽象或设计构件的特征集”。充足性从当前应用的观点来比较抽象,完备性则要考虑多个观点和询问问题,如对完整表示该问题域的对象需要什么性质。因为完备性要考虑不同的观点,它对抽象或设计构件可以被复用的程度有间接的意义。

(6) 内聚性。与传统软件的内聚性一样,一个 OO 构件应该这样设计:其所有操作一起工作,以达到单一的、定义良好的目的。一个类的内聚性通过检查它所拥有的特征集接近问题域或设计域的程度来确定。

(7) 原始性。用于操作和类,是一个操作的原子程度,即操作不能由包含在类中的其他操作序列构成。一个高原始性的类只封装原始操作。

(8) 相似性。两个或多个类,在其结构、功能、行为或目的等方面显示它们相似的程度。

(9) 易变性。当修改需求或修改系统的其他部分时,可能需要进行修改设计,从而导致设计构件的强制性修改,一个 OO 设计构件的易变性即是对修改发生可能性的度量。

关于这些设计特征度量的详细资料,可阅读 Whitmire 的《Object-Oriented Design Measurement》一书[①]。

---

① Whitmire S.. Object-Oriented Design Measurement. Wiley, 1997

15.3 OO项目的度量

项目管理人员的任务是计划、协调、跟踪和控制软件项目的进行。其中第一个活动就是计划，而早期的计划任务之一是估算。项目管理者在计划过程中面临的主要问题之一就是如何对软件的实现规模进行估算。Lorenz 和 Kidd 提出了以下几个用于 OO 项目的度量。

1. 场景脚本的数量(Number of Scenario Scripts，NSS)

场景脚本是一个详细的步骤序列，用于描述用户和应用系统之间的交互，类似于第 12 章中的用例。系统的规模及测试用例数量都与场景脚本的数量相关。场景脚本或用例的数量与满足需求所需的类的数量、每个类的状态数量以及方法、属性、协作的数量成正比。

2. 关键类的数量(Number of Key Classes，NKC)

关键类是在面向对象分析的早期进行定义的，是问题域的核心，也称为分析类。关键类一般集中在问题的业务域，并且通过复用实现的可能性很小。因此，NKC 的值表明了实际开发的工作量。Lorenz 和 Kidd 指出，在一般的 OO 系统中，关键类的比重应该在 20%～40%，其他的类支持底层结构(GUI、通信、数据库等)。

3. 支持类的数量(Number of Support Class，NSC)

支持类是实现系统所必需的，但又不与问题域直接相关的类，如 UI 类、操作类、接口类、计算类、数据库访问等。对每个关键类都可以开发其支持类。支持类的数量也是开发软件所需工作量的指标，同时也是潜在的复用数量的指标。

4. 每个关键类的平均支持类数量

关键类通常在项目早期就可以确定，而支持类的定义贯穿于项目的全过程。Lorenz 和 Kidd 指出：在采用 GUI 的应用中，支持类是关键类的 2～3 倍；在不采用 GUI 的系统中，支持类是关键类的 1～2 倍。知道每个关键类的平均支持类数量，更容易对类的总数进行估算。

5. 子系统的数量(Number of SUB-systems，NSUB)

子系统是实现某个功能的类的集合。子系统确定以后，项目组可以更容易地制定出合理的进度计划，并将子系统的工作在项目人员之间进行分配。NSUB 能为资源分配、进度安排(特别强调平行开发)和整体集成的工作量提供更好的考察。

NSS、NKC、NSUB 的度量可以用来收集过去的 OO 项目，以及整个项目或单个过程获得工作量相关的花费。这些数据可以与前面讨论过的设计度量一起用以计算生产率度量，如每个开发人员开发类的平均值、每个人月开发方法的平均值等。还可以用来估算当前项目工作量、开发时间、使用人员和其他项目信息等。

15.4　面向类的度量

经过近年来的研究，人们已经根据面向对象系统的特点提出了一系列面向对象度量，有的得到了广泛的认可和应用。这些度量分别从面向对象系统本身和系统中单个类的角度进行度量，其中较有名的是CK度量集和MOOD度量集等。

15.4.1　CK度量套件

CK度量套件是由Chidamber和Kemerer提出的，他们建议使用6种基于设计的度量[①]。

(1) 每个类的加权方法(Weighted Methods per Class，WMC)

类的加权方法数反映了类的复杂性。一般来说，类的方法数越多，类越复杂。由于类中不同方法的复杂程度不一致，所以不能用单纯的方法数相加来衡量类的复杂程度，而需要把每个方法赋予一个权值。$WMC=\sum C_i(i=1,2,\cdots,n)$，其中$C_i$为该类中第$i$个方法(或操作或服务)的复杂性，相当于传统方法中的环路复杂性。如果假设C_i都为1，则WMC就为该类的方法数。方法的数量及复杂性是实现和测试一个类工作量总量的指标。方法数量越多，继承树就越复杂，对于一个给定的类，随着方法数量的增大，其应用很可能变得越来越专门化，这就限制了它们潜在的复用，因此，WMC应当保持合理的低值。

对于一个有m个类的系统而言，其WMC是m个类的类加权方法的平均值，即

$$WMC=\frac{\sum_{j=1}^{m}WMC_j}{m} \tag{15-1}$$

其中，WMC_j为第j个类的加权方法。

这种方法乍看起来似乎很简单，每个类的方法的数量可以很直观地算出。但实际上，对于整个系统而言，如何计算方法的数量是一个比较复杂的问题，一个方法是否只属于定义它的类C，或是也属于直接或间接继承类C的类。

(2) 继承树的深度(Depth of the Inheritance Tree，DIT)

DIT定义为从结点到根的最大长度，即对象所属类在继承树中的深度，从继承的角度反映了类的复杂性。DIT越大，复杂性也越大。一般来说，DIT越大，它从父辈类中继承下来的属性和方法就越多，类的复杂性就越大。随着DIT的增大，低层次的类可能会继承很多方法，其行为的预测变得困难；但是当DIT值很大时，又意味着很多方法被复用。

(3) 子女的数量(Number of Children，NOC)

在类层次中，直接从属于某类的子类称为该类的子女。类的直接子类个数反映了该类的复用程度。随着子女数量的增加，复用也增加；但父类的抽象表示可能减少，即子女中可

① Chidamber S. R., Kemerer C. F.. A Metrics Suite for Object-Oriented Design. IEEE Trans. Software Engineering，1994，20(6)：476～493

能有一些不是父类的真正成员。而且随着 NOC 的增大，针对每个子女的测试数量也增加了。

（4）对象类之间的耦合(Coupling Between Object Classes,CBO)

CBO 指的是类之间合作的数量。CRC 模型可被用以确定 CBO 的值，CBO 的值是 CRC 卡片上列出的类的协作的数量。类之间的耦合不利于系统的模块化设计，类的耦合度越大，独立性越小，越容易受系统中其他部分的影响，越不易于复用。所以当 CBO 增大时，类的可复用性可能会降低，而且使修改及修改后的测试变得更为复杂。因此，每个类的 CBO 值应当保持合理的低值，这与之前讲的低耦合的一般原则是一致的。

（5）对类的响应(Response For a Class,RFC)

类的响应集合从另一个方面度量类的复杂度，它是所有可能被该类调用的方法的集合，执行它可以响应接收到的类对象的消息。RFC 定义为响应集中方法的数量。当 RFC 增大时，测试序列增大，测试工作量也增加了，类的总的设计复杂度也随之增大。

（6）方法中内聚的不足(Lack of Cohesion in Methods,LCOM)

类的内聚性表明了类中所有元素相互联系的程度，内聚性的高低表明对类所进行的封装是否合理。设计合理的类中的元素之间的关系应该比较紧密，因此内聚性要求较大。一个类中的每个方法可能会访问一个或多个属性，LCOM 是访问一个或多个相同属性的方法的数量。如果没有方法访问相同的属性，则 LCOM＝0。如果一个类中共有 10 个方法，其中 6 个方法共用一个或多个属性，则 LCOM＝6。如果 LCOM 的值比较大，说明该类中的某些方法通过属性与其他方法耦合，缺乏内聚的程度大，类中的元素关联程度较低，对类进行的封装可能具有不合理处，增加了设计的复杂性，通常把它分为两个或多个独立的类。人们总是希望 LCOM 值比较低，这与传统软件中的高内聚、低耦合是一致的。

下面是 CK 套件度量的例子。

【例 15-1】 图 15-1 是 ATM 系统的部分类视图，对该系统进行类的加权方法计算(假定方法的复杂度都为 1)。

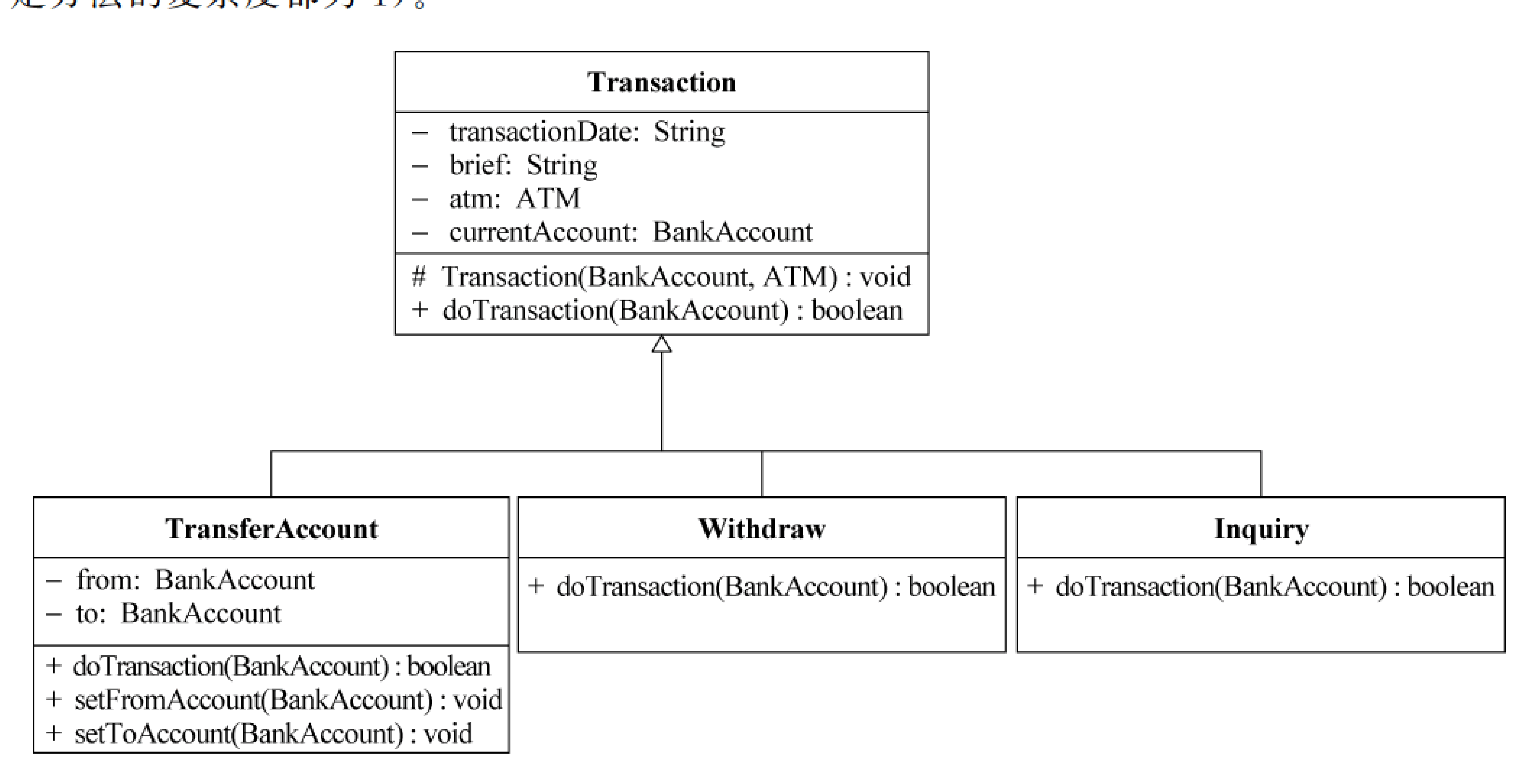


图 15-1 ATM 系统的部分类视图

解：对各类的加权方法计算如表 15-1 所示。

表 15-1　ATM 系统部分类的加权方法计算

类　名	方法	类　名	方法
Transaction	2	Withdraw	1
TransferAccount	3	Inquiry	1

$$WMC=7/4=1.75\ 方法/类$$

【例 15-2】　图 15-2 是 ATM 系统的部分类视图，对该系统进行类的耦合的计算。

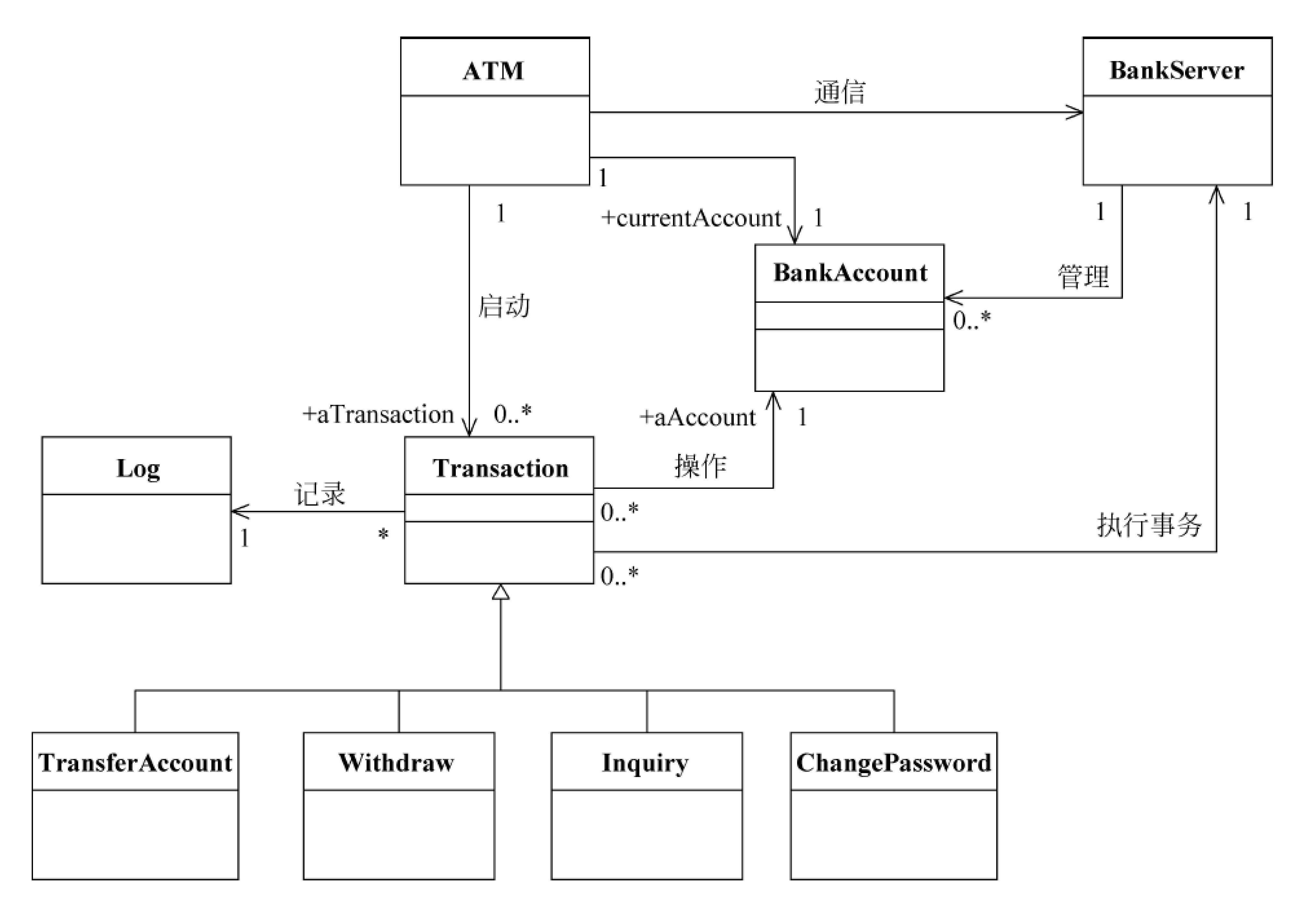


图 15-2　ATM 系统的部分类视图

解：对各类的耦合计算如表 15-2 所示。

表 15-2　ATM 系统部分类的耦合的计算

类	耦　合　类	CBO
ATM	BankServer，BankAccount，Transaction	3
BanServer	ATM，BankAccount，Transaction	3
BankAccount	ATM，BankServer，Transaction	3
Transaction	ATM，BankServer，BankAccount，Log	4
Log	Transaction	1

【例 15-3】 图 15-3 表示 ATM 系统部分类之间的协作，对该系统进行类的响应的计算。

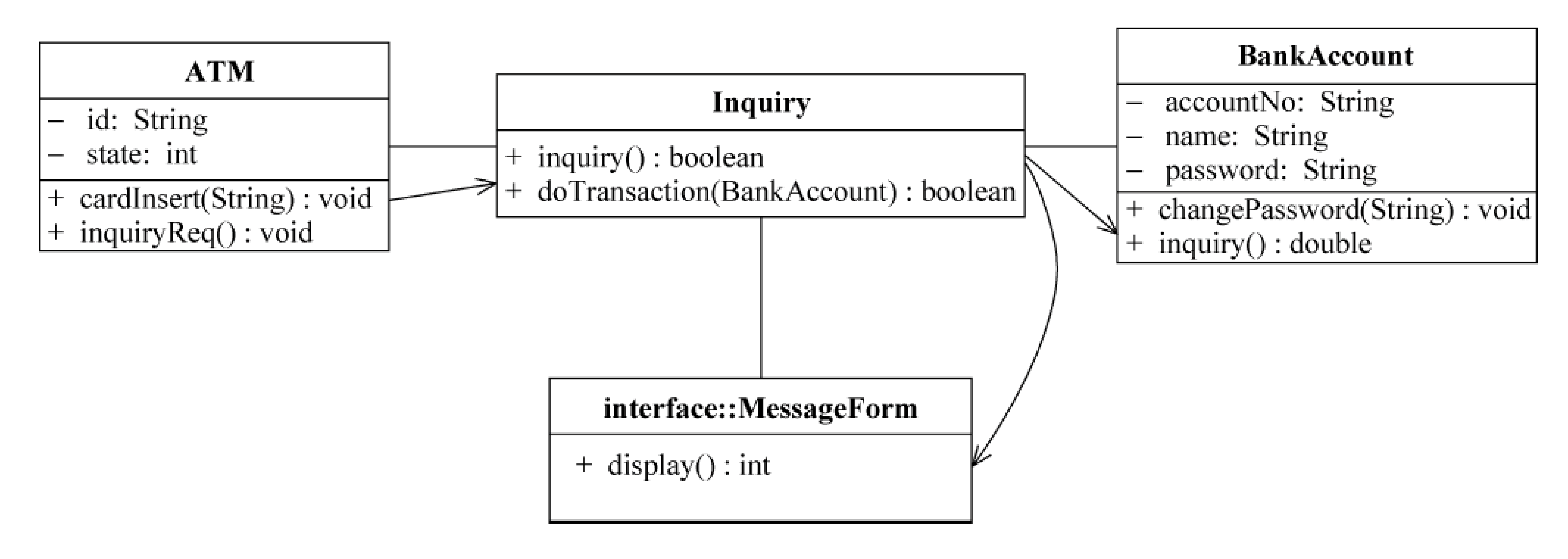


图 15-3 ATM 系统类之间的协作

解：对各类的响应的计算如表 15-3 所示。

表 15-3 ATM 系统部分类的响应的计算

类	响 应 集 合	RFC
ATM	ATM，cardInsert，inquiryReq，Inquiry，doTransaction	5
Inquiry	Inquiry，inquiry，doTransaction，MessageFrom，display，BankAccount	6
MessageForm	MessageForm，display	2
BankAccount	BankAccount，changePassword，inquiry	3

15.4.2 LK 度量组

LK 度量组是由 Lorenz 和 Kidd 提出的，他们把基于类的度量分为 4 种类型：规模、继承、内部特性和外部特性①。

1. 类的规模(Class Size，CS)

类的规模可以用以下度量确定。

(1) 在类中被封装的操作(继承和私有实例的操作)的总量。

(2) 在类中被封装的属性(继承和私有实例的属性)的总量。

CK 套件中的 WMC 实际上也是类的规模的度量。与 WMC 类似，如果 CS 的值比较高，说明一个类有很多功能，这就会减少类的可复用性，并使实现和测试复杂化。因此，继承或公有操作和属性应当增加其权值，而私有操作和属性应当更局域化、专门化，以减小实现和测试的复杂性。

2. 由子类覆盖的操作数量(Number of Operating Overridden by a subclass，NOO)

有的子类，为了自身需要，用自己的特定版本代替从父类继承的操作，这称为覆盖。

① Lorenz M.，Kidd J.. Object-Oriented Software Metrics. Prentice-Hall，1994

NOO 的值越大，说明子类覆盖的操作越多。Lorenz 和 Kidd 指出，子类应该是超类的一个特例化，它应该主要扩展超类的服务(操作)。因此，较高的 NOO 可以提高类的复用性，并减少实现和测试的复杂性；但是如果 NOO 过大，则违反了超类所蕴涵的抽象，削弱了类的层次结构，使测试和修改的难度增大。

3. 由子类增加的操作数量(Number of Operating Added by a subclass，NOA)

子类通过加入私有操作或属性实现实例化。当 NOA 的值增大时，则子类漂离超类所隐含的抽象。当类层次的深度增大时，在低层的类的 NOA 的值将下降。

4. 特例化指标(Specialization Index，SI)

特例化指标为 OO 系统中的每一个子类提供了特例化等级的粗略指示，可以通过增加或删除操作或覆盖实现。

$$SI = [NOO \times level]/M_{total} \tag{15-2}$$

其中：level 表示类在类层次中的层数，M_{total} 表示类方法的总数。

SI 的值越大，类层次中包含了不遵从超类抽象的类的可能性也就越高。

15.4.3　MOOD 度量套件

MOOD 度量套件是由 Harrison、Counsell 和 Nithi 提出的，是 OO 设计特征的一组量化指标，它从整个面向对象系统的角度对系统的面向对象特征进行评估。以下给出 MOOD 套件的部分度量公式①。

1. 方法继承因子(Method Inheritance Factor，MIF)

MIF 是一个 OO 系统类结构中对方法(操作)和属性使用继承的程度，定义如下：

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

其中：TC 为体系结构中类的总数；C_i 为体系结构中的一个类；$M_a(C_i) = M_d(C_i) + M_i(C_i)$，$M_a(C_i)$ 为与 C_i 相关联的被引用的方法的数量，$M_d(C_i)$ 为类 C_i 中声明的方法的数量；$M_i(C_i)$ 为类 C_i 中继承(没有被覆盖)的方法的数量。由 MIF 的计算方法可以看出，方法继承因子是一个 0～1 之间的值，它的值越大，表明系统中的方法继承的程度越高。

另外还有属性继承因子(Attribute Inheritance Factor，AIF)，与 MIF 的定义类似。MIF 和 AIF 提供了方法和属性继承对 OO 软件影响的量化指标。

2. 耦合因子(Coupling Factor，CF)

耦合因子 CF 是对 OO 设计元素之间关联的指标，一般情况下，认为在两个类之间有任

---

① Harrison. R.，Counsell S. J.，Nithi R. V.. An Evaluation of the MOOD Set of Object-Oriented Software Metrics. IEEE Trans. Software Engineering，1998，24(6)：491～496

何形式的消息传递，它们就存在耦合关系。CF 的定义如式(15-3)所示。

$$CF = \frac{\sum_{i=1}^{TC}\sum_{j=1}^{TC} is_client(C_i, C_j)}{TC^2 - TC} \tag{15-3}$$

其中：函数 $is_client=1$，当且仅当客户类 C_c 和服务器类 C_s 间存在关联，且 $C_c \neq C_s$；否则 $is_client=0$。显然，CF 的值在 0～1 之间，当 CF 值增大时，类之间的关联增多，OO 软件的复杂性也将增加，软件的可理解性、可维护性及可重用性也将随之受到影响。

3. 多态因子(Polymorphism Factor，PF)

多态性是一个值(属性、方法)具有多于一种类型的能力，使它可以被运用到不同类型的多个语境中。重载就是多态的一种常见的形式，它允许一个函数名被多次使用，每次都带有不同类型的参数。系统在运行时，根据类的类型确定调用哪段代码。相对于其他语言，具有多态特征的语言具有很多优势，如灵活性强、抽象性好、支持一定程度的代码共享和行为共享。

多态因子 PF 表明系统中所有类的方法使用多态机制的程度，它是系统中实际发生的多态数目和可能发生的多态情况的最大数目的比值，是系统中动态结合(绑定)的相对数量的间接度量。

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC}[M_n(C_i) \times DC(C_i)]} \tag{15-4}$$

其中：$M_n(C_i)$为新方法的数量；$M_o(C_i)$为覆盖的方法的数量；$DC(C_i)$为某基类后代类数量的计数。显然有 $M_d(C_i)=M_n(C_i)+M_o(C_i)$。可以看出，PF 中考虑的多态，都是由于继承引起的，它的值在 0～1 之间，值越大，表明多态的发生越频繁。

下面是 MOOD 套件度量的几个例子。

【例 15-4】 用 MOOD 套件对下列 java 代码进行 MIF 和 PF 的度量。

```
public class Transaction {
        private String transactionDate;
        private ATM atm;
        private BankAccount currentAccount;
        public BankServer m_BankServer;
 protected void Transaction(BankAccount bankaccount, ATM atm){      }
 public boolean doTransaction(BankAcount currentAccount){      }
}

public class Inquiry extends Transaction {
 public boolean doTransaction(BankAcount currentAccount){      }
 private boolean inquiry(){      }
 }

public class Withdraw extends Transaction {
 public boolean doTransaction(BankAccount currentAccount){      }
 private boolean withdrawal(){      }
 }
```

解：MIF 和 PF 度量结果分别如表 15-4 和表 15-5 所示。

表 15-4　MIF 度量计算表

类	$M_i(C_i)$（类 i 中继承的方法的数目）	$M_d(C_i)$（类 i 中声明的方法的数目）
Transaction	无	Transaction()，doTransaction()
Inquiry	Transaction：Transaction ()	doTransaction ()，inquiry()
Withdraw	Transaction：Transaction ()	doTransaction ()，withdrawal()

$$MIF=2/(2+6)=1/4$$

表 15-5　PF 度量计算表

类	$M_n(C_i)$	$M_o(C_i)$	$D_C(C_i)$
Transaction	Transaction()，doTransaction()	无	2
Inquiry	inquiry()	doTransaction ()	0
Withdraw	withdrawal()	doTransaction ()	0

$$PF=2/(2\times 2+1\times 0+1\times 0)=1/2$$

【例 15-5】　对图 15-2 所示的类图算出耦合因子 CF。

解：CF 度量计算如表 15-6 所示。

表 15-6　CF 度量计算表

序号	类	*is_client* 类
1	ATM	BankServer、BankAccount、Transaction
2	BankServer	BankAccount
3	BankAccount	无
4	Transaction	BankServer、BankAccount、Log
5	Log	无

$$CF=7/(5^2-5)=7/20$$

15.5　面向操作的度量

类是 OO 系统中最基本的单元，系统的连接性结构可能比个体模块的内容更为重要，但是，通过检查类操作的一般特征也可以帮助我们更好地度量系统。根据 Lorenz 和 Kidd 的建议，有以下 3 种简单的度量。

1. 平均操作规模（average Operation Size，OS_{avg}）

虽然代码行 LOC 可以作为操作规模的指标，但是 LOC 测量存在依赖于程序设计语言，不适用于非过程语言等问题，因此，在 OO 软件中，操作传送的消息的数量成为操作规模的另一个度量方法。操作规模增大，表明操作所传送的消息增多。OS_{avg} 值越大，说明操作越复杂，责任在类中没有很好地分配。

2. 操作复杂度(Operation Complexity,OC)

操作的复杂度可以用传统方法中所使用的任何复杂性度量进行计算。由于操作要实现特定的责任,应该使 OC 的值尽可能小。

3. 每个操作所带参数的平均数量(average Number of Parameters per operation,NP_{avg})

操作参数数量越多,对象间的合作越复杂,所以 NP_{avg} 的值要尽可能小。

15.6 面向对象测试的度量

在面向类和面向操作的度量中给出设计质量的指标,同时也是计算 OO 系统测试工作量的一般指标。

Binder 提出了一组设计度量,对 OO 系统的可测试性有直接的影响。这组度量被组织为能反映重要设计特征的类别。

(1) 封装

① 方法中内聚的不足(LCOM)。LCOM 值越大,说明该方法与其他方法的耦合度越高,为了保证该方法不会产生副作用,必须测试的状态也就越多。

② 公共和私有属性的百分比(percent Public And Protected,PAP)。公共属性是继承来的,对某些类来说是可见的。而私有属性是特例化的,为特定的类所私有。PAP 给出了公共属性所占的百分比,值越高,说明公共属性越多,类之间副作用的可能性也增加了。为此,必须设计测试,以保证能发现这样的副作用。

③ 对数据成员的公共访问(Public Access to Data members,PAD)。PAD 是某个类可以访问另一个类的属性(或方法)的数量,这实际上违背了封装的原则。PAD 的高值同样导致了类之间副作用的可能,必须设计测试以保证发现这样的副作用。

(2) 继承

① 根类的数量(Number Of Root classes,NOR)。NOR 是在设计模型中描述类层次的数量,要为每个根类和对应的类层次设计测试用例。随着 NOR 的增加,测试工作量也随之增加。

② 扇入(Fan IN,FIN)。在 OO 语境中,FIN 是多继承的指标。FIN>1 表明该类从多于一个的根类继承属性和操作,这种情况应该尽可能避免。

③ 子类的数量 NOC 和继承树的深度 DIT。在面向类的度量中介绍了 CK 套件,NOC 和 DIT 是其中的两个度量。前面也已经介绍过了,随着 NOC 和 DIT 值的增大,增加了复用;但是系统变得更为复杂,超类的方法必须针对每个子类被测试,测试数量也要随之增加。

除了上面的封装和继承度量以外,Binder 还定义了类的复杂性和多态性的度量。类的复杂性度量包括 CK 套件中的每个类的加权方法 WMC、对象类之间的耦合 CBO 和对类的响应 RFC,还定义了和方法计数相关的度量。多态性的度量高度专门化,这里就不再进行讨论了。

本 章 小 结

度量可以改善软件过程,辅助软件项目的计划、跟踪及控制,评估软件质量。对 OO 系统的度量着重于度量可以应用于类的类所特有的设计特征——局部化、封装、信息隐蔽、继承、抽象等。

Whitmire 提出了关于 OO 设计的分析、设计模型的独特的可测度的 9 个特征，有助于 OO 系统的项目管理者的计划和跟踪。场景脚本、关键类、支持类、子系统的数量等度量值提供了实现系统所需的工作量的信息。

面向类的度量主要介绍了 CK 度量集合 MOOD 度量集。CK 建议使用 6 种基于设计的度量，侧重于类和类的层次；MOOD 度量则是对系统的面向对象特征进行评估。

面向操作的度量着重介绍个体操作的大小和复杂度，其切入点在类级别。

本章还对 OO 系统的可测试性进行了评估，侧重于封装、继承等。

思考与练习

1. 针对传统软件的度量和 OO 软件的度量有何不同？
2. 面向对象系统的 5 个特征如何影响传统软件及 OO 软件的度量？
3. 设类 X 有 5 个操作，其复杂度分别为 2、5、8、11、4，计算 X 的 WMC。
4. 计算图 15-4 中每棵继承树 DIT 的值；在两棵树中，类 X2 的 NOC 分别是多少？

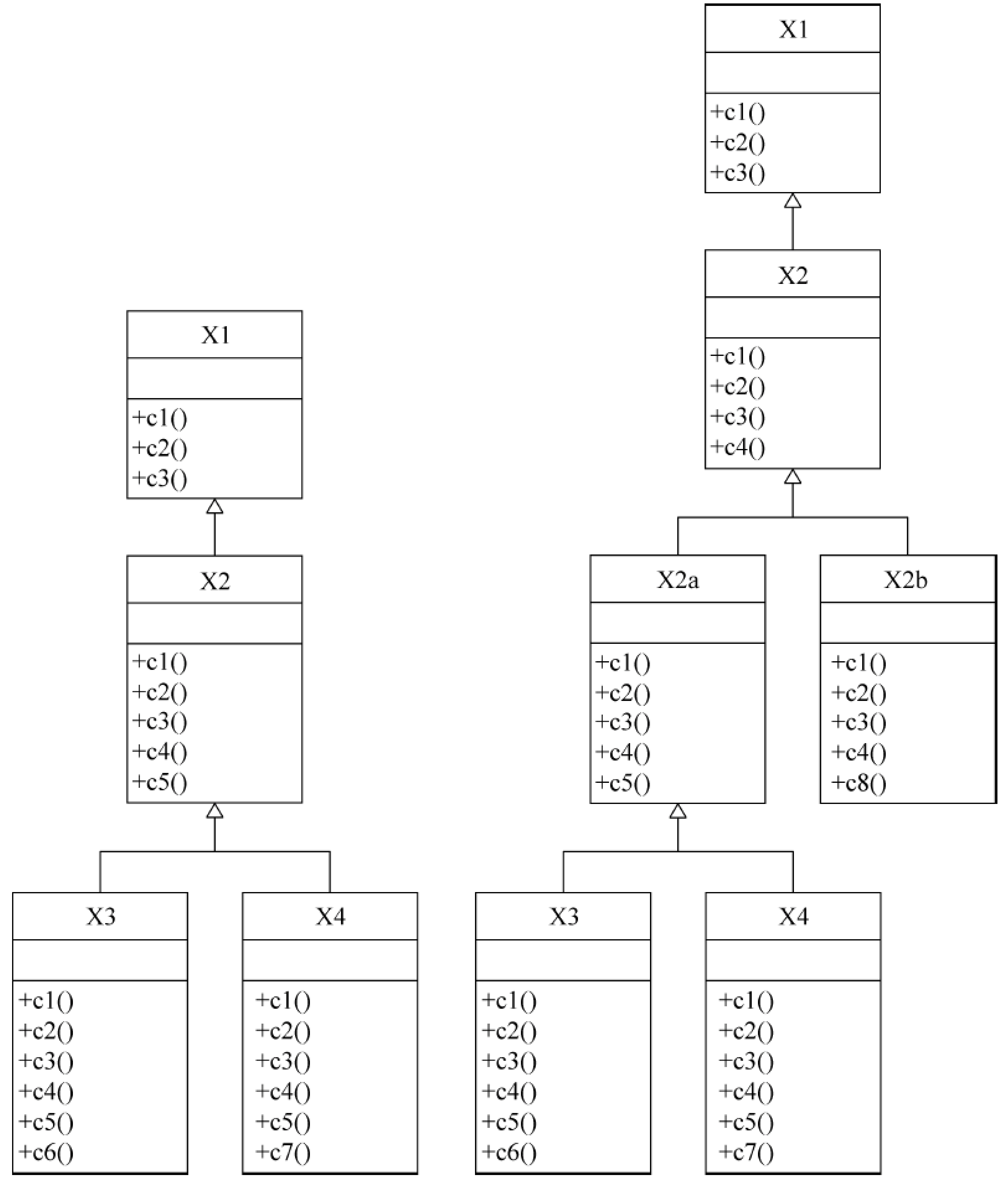


图 15-4　两棵树

5. 对图 15-4 中两棵树而言，类 X3 和 X4 的 NOA 分别是多少？

6. 图 15-5 表示了简化的图书借阅系统的类视图，对该系统进行基于 CK 套件的类的加权方法以及对象类之间的耦合的度量。

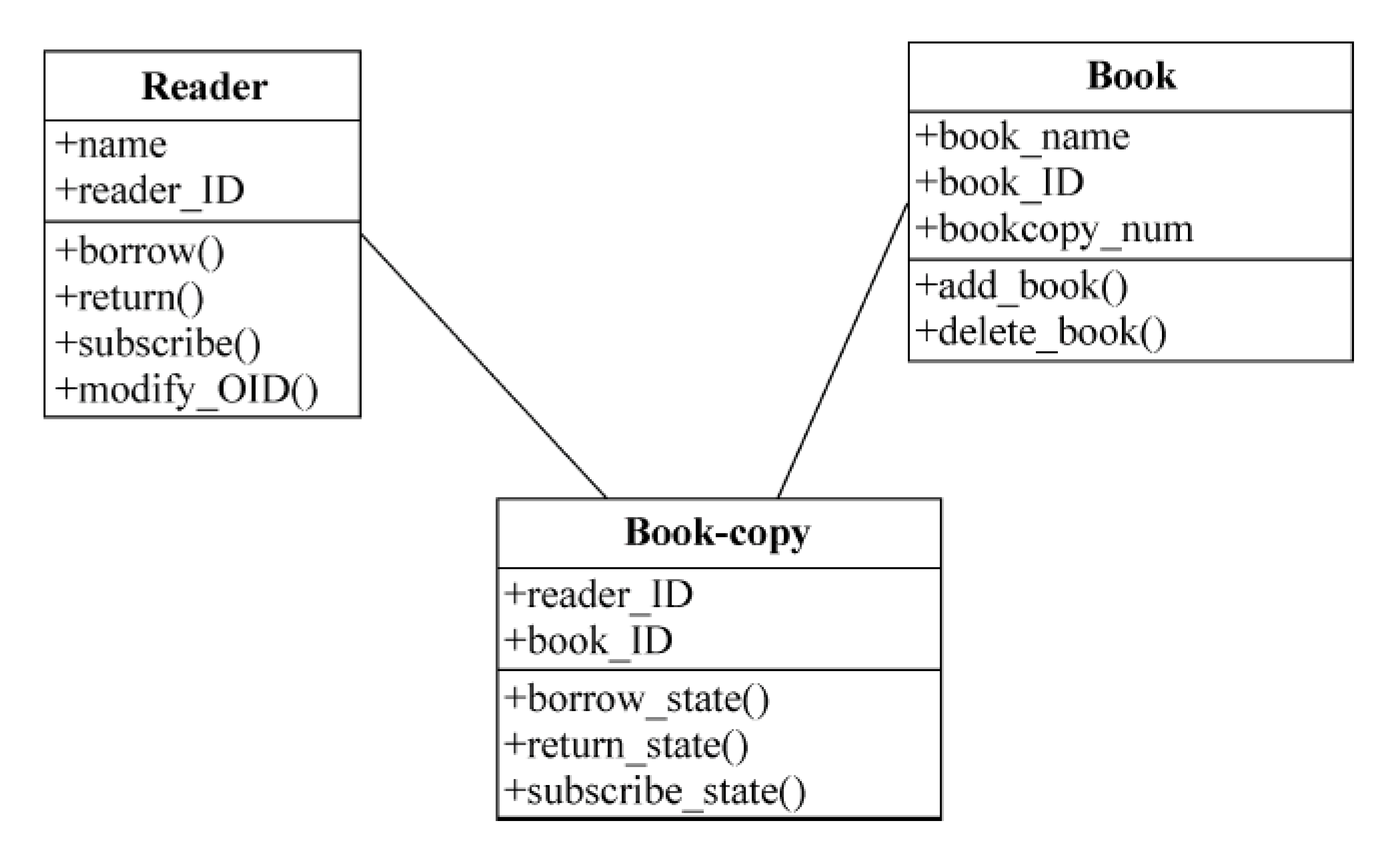


图 15-5 简化的图书借阅系统的类视图

7. 对如图 15-6 所示的图书借阅系统的类之间的函数调用图进行类的响应（RFC）的度量，图中的箭头表示每个函数调用（仅给出类间调用）其他类的函数。

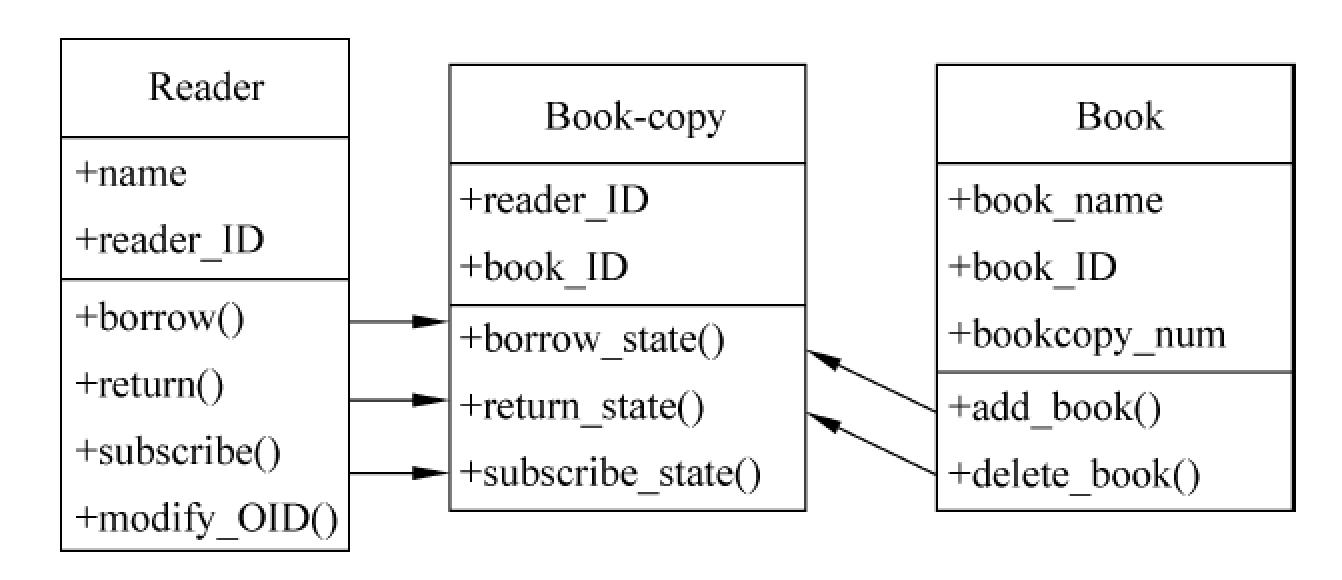


图 15-6 图书借阅系统类之间的函数调用图

8. 用 MOOD 套件对下列 C++代码进行 MIF 和 PF 度量。

```
Class A{
      protected:
      int a;
      public;
      void x();
      virtual void y();
};
Class B: : public A{
      Int b;
      protected;
      int bb;
      public void w();
      public void y();
      public void z();
};
Class C: : public B{
```

```
    int c;
    void v();
};
```

9. 根据图 15-7 算出耦合因子 CF。

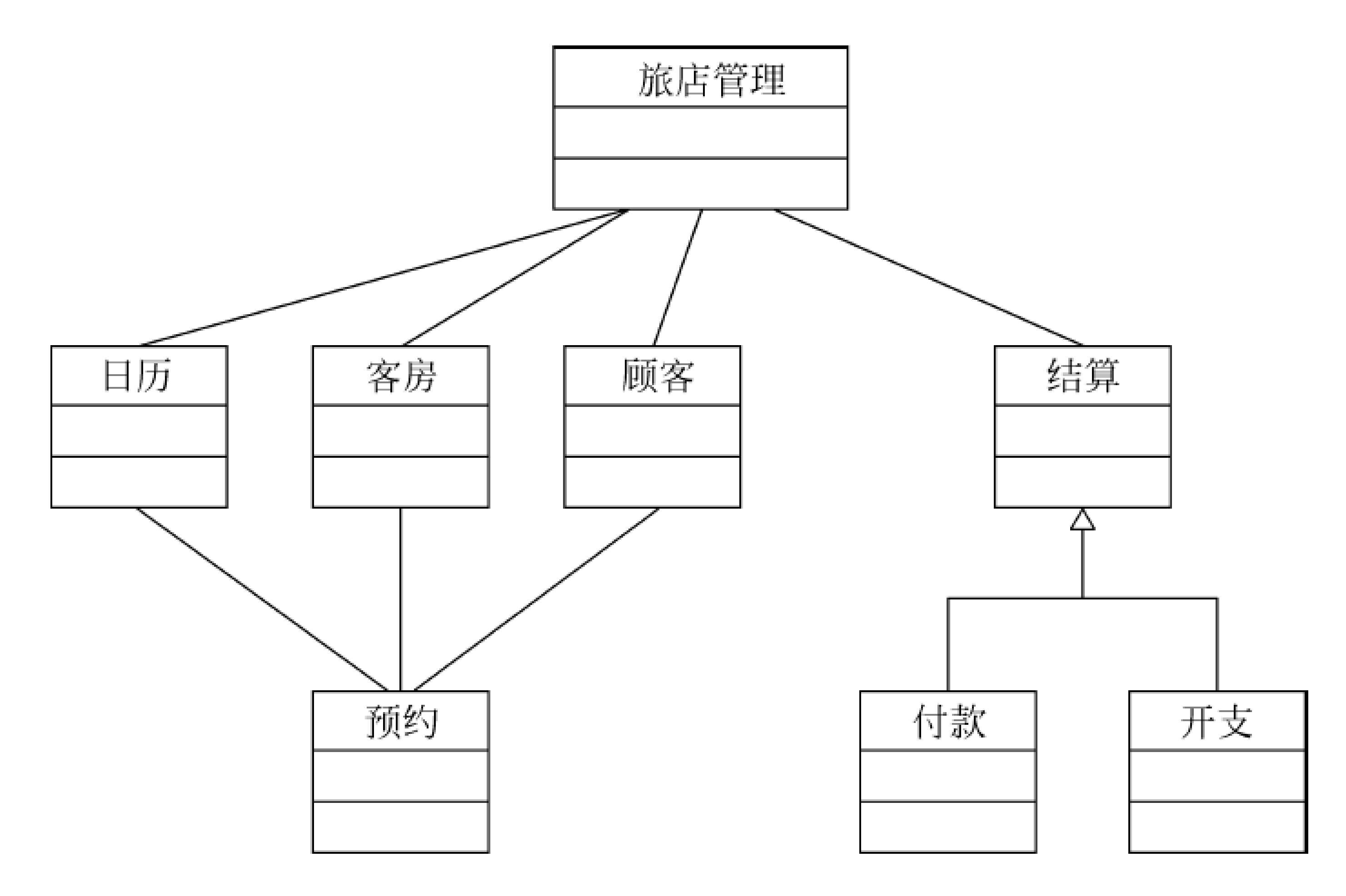


图 15-7　旅店管理系统示意图

第4篇 软件工程高级专题

本篇内容：

- 敏捷过程开发
- Web工程
- 形式化方法



第16章 敏捷过程开发

唯一不变的是变化本身。

——SpencerJohnson(美国著名作家、思想家)

2001年2月,软件过程方法论学者峰会接受了敏捷方法的定义,并形成了《敏捷软件开发宣言》[①],目的在于说明什么是敏捷软件开发。该宣言声明:

我们正在通过亲身实践以及帮助其他人实践,揭示更好的软件开发之路,我们认为:

个体和交互胜过过程和工具

能够发挥作用的软件胜过面面俱到的文档

客户合作胜过合同谈判

随时应对变化胜过遵循计划

也就是说,虽然我们也关注上述右边的内容,但是更关注左边的内容。

《敏捷宣言》是一份寻求反传统软件开发观点的非常简明的声明。它的基础是以下12条原则:

我们的首要任务是通过尽早并且持续提供有价值的软件满足客户的需要。

即使在开发后期,也欢迎需求变更。敏捷过程驾驭变更,使客户获得竞争优势。

频繁提供能够发挥作用的软件,间隔时间从几周到几个月,时间间隔越短越好。

在整个项目期间,业务人员和开发人员每天都必须在一起工作。

围绕士气旺盛的人进行软件开发,向他们提供所需的环境和支持,相信他们能够完成任务。

向开发团队和在开发团队内部传递信息最高效、最有效的方法,就是面对面的交流。

能够发挥作用的软件是工作进展的主要度量标准。

敏捷过程提倡可持续开发。出资方、开发方和用户都应该能够保持一种长期、稳定的开发速度。

对卓越技术和良好设计的不断追求可提高敏捷性。

简单——尽可能少的工作量是至关重要的。

最好的体系结构、需求和设计都来自于自组织团队。

团队定期反思如何提高有效性,并相应地调整自己的行为。

---

① David J. Anderson. 软件工程的敏捷管理. 韩柯等译. 北京：机械工业出版社,2004

从本质上讲,敏捷方法是为了克服传统软件工程中认识和实践的弱点开发而成的,它提出了一些不同寻常的工作实践,引入了令人恐慌的违反直觉的工作实践。敏捷开发可以带来多方面好处,但它并不适用于所有项目、所有方面、所有人和所有情况,不完全对立于传统软件工程实践。

16.1 敏捷的定义

16.1.1 什么是敏捷

敏捷性通常定义为"应对变更的能力"[①]。根据 Darwinian 的定义,敏捷性是一种"遗传适应性"。遗传适应性是对应对不断变化的环境、进化并避免灭亡的物种能力的一种度量。大多数敏捷学者都没有采用这种方法定义敏捷性,他们把软件开发看做是一种大体上是混沌(不可知)世界中的遗传现象,认为敏捷方法不应该寻求策划或控制,而应该允许出现关注和度量某个整体目标的自组织行为,而这种目标永远都是交付最终可执行代码。事实上,这只是描述敏捷方法能够应对变更的一种行为观察方式,它没有进行策划,而是允许"第一线开发人员"做出决策。这类方法具有灵活机动的性质,因此能够应付变更。

那么在软件工程这个环境下,敏捷是什么? Ivar Jackson 给出一个有用的论述[②]:

敏捷已经成为当今描述现代软件过程的时髦用词。每个人都是敏捷的,敏捷团队是能够适当响应变化的灵活团队。变化就是软件开发本身,软件构建有变化、团队成员在变化、使用新技术会带来变化,各种变化都会对开发的软件产品以及项目本身造成影响。我们必须接受"支持变化"的思想,它应当根植于软件开发中的每一件事中,因为这是软件的心脏和灵魂。敏捷团队意识到软件是由团队中所有人共同开发完成的,这些人的个人技能和合作能力是项目成功的关键所在。

在 Jackson 的观点中,普遍存在的变化是敏捷的基本动力,软件工程师必须加快步伐以适应 Jackson 所描述的快速变化。但是,敏捷不仅仅是有效地响应变化,它还应该包含对本章开头所提宣言中哲学观念的信奉。它鼓励能够在组员之间、技术和商务人员之间、软件工程师和经理之间进行更便利沟通的团队结构和协作态度,强调可运行软件的快速交付而不是中间产品,将客户作为开发组成员以消除普遍存在于多数软件项目中的"区分你我"的态度,意识到在不确定的世界里计划是有局限性的,必须是可调整的。

敏捷可用于任何软件过程,实现要点是将软件过程设计成如下方式:允许项目团队调整并合理安排任务,理解敏捷开发方法的易变性并制定计划,精简并维持最基本的工作产品,强调增量交付策略,快速向客户提供适应产品类型和运行环境的可运行软件。

---

① Jackson I.. A Resounding "Yes" to Agile Processed-But Also More[J]. Cutter IT Journal,2002,15(1):18～24

② Jackson I.. A Resounding "Yes" to Agile Processed-But Also More[J]. Cutter IT Journal,2002,15(1):18～24

16.1.2 什么是敏捷过程

敏捷不是一个过程，是一类过程的统称，它们有一个共性，就是符合敏捷价值观，遵循敏捷的原则。敏捷过程能持续地适应：①源自开发过程中获取的经验而进行的变更；②软件需求的变更；③开发环境的变更。

敏捷过程特别支持尽早尽快地交付可工作代码的产品，这是通过迭代的开发过程完成的，其中每次迭代都注重提交可工作的代码以及其他产品以供客户评估，同时也供项目评估。敏捷过程的支持者和批评者都强调在这些过程中注重代码。支持者经常争论说代码是唯一重要的可交付的产品，可以忽视分析和设计模型、文档在软件开发、演化过程中的角色。敏捷过程批评者指出，强调代码能带来全体记忆丢失(Corporate Memory Loss)，因为没有重视编写良好的文档和模型来支持庞大、复杂软件系统的创造和演化。敏捷支持者和批评者提出的声明引出这样的问题：在当今快速变化的开发环境中，什么样的实践、技术和基础结构适合软件开发过程?

敏捷过程强调在实践中协作，目前的敏捷过程提供了一个过程框架限制开发人员必须遵守的过程形式。例如，大多数发布的在敏捷过程上的作品规定迭代的、增量的过程，并且提倡诸如先编写测试代码、结对编程和每日审查会议等特殊形式的实践。

敏捷相对以前的软件工程最大的革新之处在于把人的作用提高到了过程之上，涉及人的问题，就已经不再是过程所能覆盖的，就到了企业管理的层面上，包括企业的价值观和文化。这也是敏捷在国内实施的最大障碍。

(1) 把客户当作合作伙伴而不是对手，从客户角度出发去想问题，充分地跟客户沟通，而不是出了问题推诿责任。目标是让软件实现客户的价值，而不是收完钱就结束了。

(2) 把人的能动性调动起来，给动力而不是给压力。

(3) 要实用而不是要规范。让开发人员理解并实施，体验到敏捷的好处，而不是盲目机械地实施规范。

(4) 没有绝对的权威，每个人都有可取之处。

16.2 敏捷过程模型

下面将简要介绍几种敏捷过程模型，这些方法有相似之处，我们将尝试着通过强调每种方法的特点来展现其与众不同之处。每种敏捷方法(或多或少)都遵循敏捷软件开发宣言及其原则。

16.2.1 极限编程

极限编程(eXtreme Programming，XP)使用面向对象方法作为推荐的开发范型，包含了策划、设计、编码和测试4个框架活动的规则和实践[①]。XP中的思想及实践应用都不是新的东西，但其思想基础和组合方式是新的，极大地改善了“古老的”实践。

① Roger S. Pressman. 软件工程——实践者的研究方法. 郑人杰，马素霞，白晓颖等译. 北京：机械工业出版社，2008

XP 最早是打算供小型、紧密协作的团队使用的，它的最简化方法很著名。从某种意义上说，XP 方法看上去像规则，规定哪些能做哪些不能做。

1. 策划

策划活动开始于建立一系列描述待开发系统必要特征和功能的“场景”，一个场景定义一个特定的特性需求并在一个简单的卡片上显示。客户根据对应特征或功能的全局业务价值确定场景的权值(优先级)，当然还要考虑风险和复杂度。

短期(3 周以内)的迭代，不定期的计划改进和分派“场景”都是 XP 要策划的内容。每个发布要尽可能小，并包含大多数有价值的业务需求，这样可以提供在大项目中少有的对成果的感性认识，以及经常的、相关的反馈。一旦某个场景的开发成本超过了 3 周，应该把该场景进一步细分，重新赋予权值并计算成本。客户和 XP 团队共同决定如何把场景分组并置于 XP 团队将要开发的下一个发布版本中。因此，策划阶段的发布计划要标明到产品发布日期之前分派到各迭代中的场景。XP 团队对待开发的场景进行排序，有 3 种方式：①所有选定场景将在几周内尽快实现；②具有最高价值的场景要移到进度表前面并首先完成；③高风险场景将首先实现。

2. 设计

XP 设计包括两方面内容：①设计已经定义的功能，而不是开发者假定将来可能会用到的功能；②以尽可能简单的方式来设计要交付的功能。XP 鼓励使用 CRC 卡片来组织和当前软件增量相关的对象和类。

XP 方法运用重构来弥补体系结构、分析和设计中前期工作的不足。所有的开发人员和管理者都明白，改变已存在的代码是一件很痛苦的事。而**重构**就是在编码完成后改进代码设计，它是以不改变代码外部行为而改进其内部结构的方式来修改软件系统的过程。当需要增加新功能时，第一步通常是为了简化新功能加入而重构，被提议的新功能提供了进行重构的动力。因此，重构往往被认为是增量的，而不是大量的再设计。在理想情况下，重构应该不断发生，并且对外部是透明的。流程是通过连续重构实现的。

3. 编码

XP 的编码活动采用**结对编程**的方式。结对编程寻求充分利用开发人员这种软件生产系统中的能力受限资源。在很少的几种几乎被全面接受并得到极大认同的软件工程实践中，软件检查是其中之一。最好的情况是检查可以加速学习所发现缺陷的交互协作；而最差情况，检查降格为伴随着虚假表格和度量标准的人身攻击会议。Kent 认为“结对编程是两个人之间试图同时编程和如何编得更好的对话”①。当两个人同时坐在一台计算机前，一个进行编码或案例测试，另外一个评审和思考，创造一个持续又动态的交互氛围。有人认为，结对编程把其他方法的代码审查过程提升到一种连续过程，把它看做是一种连续审查。

结对编程可以提高软件生产系统的整体效能，主要原因是替换和积极的同行确认。结对编程中的两个开发人员可以交换位置。软件开发是很累人的工作，如果开发人员交替地

---

① Beck Kent. Extreme Programming Explained：Embrace Change. Boston：Addison-Wesley，2000

在编辑器中输入，每天生产率高的时间会更长。积极的同行确认可以提高生产率。如果同行肯定自己做得很好，开发人员全天都会保持高昂的精神状态，使高生产率状态保持得更久。

结对编程还可以实现技能转移，提高整个团队的有效性和生产率，因此 XP 方法建议程序员要定期交换结对。

但是在结对编程的过程中，相处太好的对子可能会花太多的时间谈论与工作无关的事，从而降低工作效率；而相处不好的对子，因为两个人之间不会尽力相互帮忙，也可能是低效的。

站立会议是充分利用开发人员这个能力受限资源的另一个工具。

当开发人员遇到难题时可以提出寻求帮助，也可以提出更多的工作要求。这有助于减少窝工现象，还可以尽早发现并解决问题，避免问题造成危害。开发人员可以报告已完成的工作，如果每天都得到同伴不断的认可，可以增强开发人员的信心，提高工作热情。站立会议还可以产生来自团队其他成员对自身带来的压力，因为所有成员都期望团队表现很强，并保持下去。

4. 测试

XP 不推荐在场景策划和基本设计完成以后，直接开始编码，而是开发一系列用于检测本次发布的单元测试，即“预先测试，然后再编码”。这非常接近于“测试驱动的开发”，它们都提倡在编写代码前首先编写测试用例。由于在编码前就设计测试用例，所建立的单元测试应当使用一个可以自动实施的框架，一旦代码完成，马上交付测试，即时反馈，这种方式支持代码修改后即时的回归测试策略。

16.2.2 自适应软件开发

自适应软件开发(Adaptive Software Development，ASD)是一个基于协作的敏捷开发方法，定义了如何在高速、变更性和非确定性等强化复杂性的关键特征下开发更好的软件。ASD 是用于变化，不是抵制变化，必须拥有实践来接受并做出相应反应。ASD 的实践是由连续适应驱动的，在 ASD 中，静态的策划—建模—构建生命周期被动态的预测—协作—学习的生命周期(图 16-1)所取代。这个生命周期致力于连续学习和面向变化、再次评估、在不确定的未来中显现，以及开发人员、管理人员和客户的密切协作。自适应周期有 5 个关键特性：①自适应周期是目标驱动的；②自适应周期是基于部件而不是基于任务的，注意力要集中在细化部件的定义上面(即希望的结果)，而不是列举产生结果所需要的活动；③自适应周期是迭代的；④周期是有时间区间限制的；⑤周期是风险驱动和容变的[①]。

1. 预测：启动与规划

(1) 项目初始化，包括设置项目的任务和目标、理解约束、建立开发项目中的各种组织、识别和概括需求、制定启动规模和范围的推测、确认关键的项目风险。

---

① James A. Highsmith. 自适应软件开发. 钱岭等译. 北京：清华大学出版社，2003

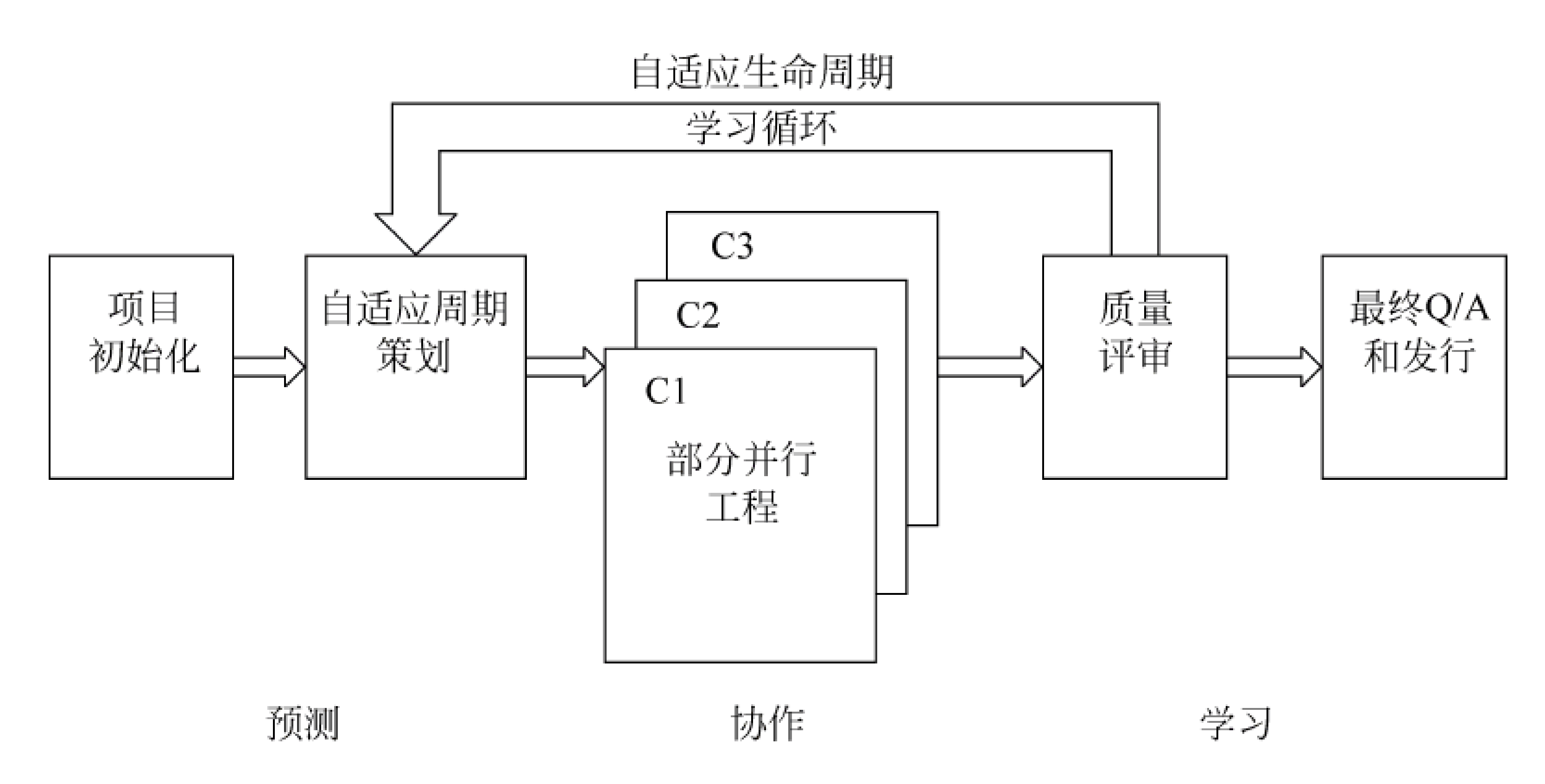


图 16-1 自适应生命周期

(2) 根据项目的范围、需求、估计,由项目初始化工作确定的资源可用性来决定整个项目的时间框。

(3) 决定迭代的次数,并制定各自的时间框。对于中小型应用,一般为 4～8 周。周期应反映出开发团队对产品需求不确定性的感觉。短周期应该用于高风险区域,长周期用于更确定的区域。

(4) 在建立迭代及其进度表后,要为每个迭代确定主题或目标。对于客户评审过程,每个迭代应当交付可演示的特征集。在迭代中,开发团队每天,甚至更频繁地将正在开发的特性进行集成。测试和特性开发一起进行,并成为特性开发不可分割的一部分。

(5) 开发人员和客户为每次迭代分配特性。在分配过程中,客户通过使用特性估计、风险和开发团队所提供的相关信息决定特性的优先级。

2. 协作:并发特性开发

协作方法是所有敏捷方法中不断重现的主旋律。协作是共享创造的行为,需要互信和互相尊重。协作离不开稳定的团队,但是也不排斥"个人主义",个人的创造力在协作中起着重要作用。团队要在技术问题、业务需求和快速决策等方面进行协作。

3. 学习:质量评审

由于自适应项目的环境是扰动和不确定的,因此需要在每个开发周期的末尾进行反思、状态确定和学习。对于线性的、瀑布型软件开发来说,不赞成每个阶段结束后回顾,而学习只有在错误和试验中进行,这需要团队成员很早就部分共享已完成的代码和产品,这样才能发现错误、相互学习、减少因为小问题变成大问题而造成的返工。在每次迭代的末期,需要学习四大类内容:以客户角度期待的结果质量、从技术角度期待的结果质量、交付产品的团队的功能和团队成员使用的实践、项目的状态。

ASD 团队通过以下 3 种方式学习,利用对评审中提出附加问题的回答来评估理解力:

(1) 客户组评审——客户对已发布的产品有什么想法,这些产品是否满足他们的业务需求和其他定义的质量标准。对已发布的软件增量提供反馈,给出产品是否满足需求的直接提示。

（2）正式技术评审——ASD 团队成员对已发布的产品有哪些想法。通过评审软件构件来提高质量、学习知识。

（3）事后剖析——ASD 团队成员对自己的表现有什么想法，实际和目标之间有什么差距。自我反省，着眼于自身的表现和过程。

16.2.3 动态系统开发方法

动态系统开发方法（Dynamic System Development Method，DSDM）是一种提供"通过在可控项目环境中，使用增量原型开发模式完全满足对时间有约束的系统的构建和维护"的敏捷软件开发方法[①]。像 XP 和 ASD 一样，DSDM 建议使用迭代软件过程。DSDM 提出了探索式开发方法的概念，强调"没有什么事能一次就做好"，强调系统使用者不可能在一开始就预见所有需求。DSDM 的每次迭代都遵循 80-20 规则，即每个增量只完成能够保证顺利进入下一增量的工作，剩余的细节可以在知道更多业务需求或提出并同意变更之后完成。

DSDM 的 9 条原理与敏捷宣言的原理是一致的[②]：

（1）用户的积极参与是必要的。

（2）必须赋予 DSDM 团队决定权。

（3）重点在于产品的经常性交付。

（4）适应业务需要是所交付产品被接受的一个基本标准。

（5）迭代和增量式开发对于最终给出精确的业务解决方案是必要的。

（6）开发期间的任何修改都是可逆的。

（7）需求必须定位在高水平上。

（8）测试必须贯穿整个生命周期。

（9）所有项目相关人员之间的协作方法是至关重要的。

图 16-2 展示了 DSDM 的开发过程，每个主要部分——功能模型迭代、设计与构建迭代以及实现迭代，都是它们各自的迭代。在 3 个迭代之前还有两个生命周期。

（1）可行性研究——建立要开发应用的业务需求和相关约束，并评估 DSDM 过程是否可行。

（2）业务研究——建立系统提供业务价值所需要的功能和信息需求，同时确定基本的系统架构，并识别软件的可维护性需求。

- 功能模型迭代。搜集和确定功能需求，为客户开发一系列证明其功能的增量原型，通过用户使用原型系统诱导反馈信息以获取额外的需求。
- 设计和构建迭代。精化原型以使其满足所有需求并使设计的软件也满足这些需求。一套业务功能在一个时间框中可能通过功能模型、设计与构建的迭代，继而另一套功能在下一时间框内可以通过同样的过程。有些情况下，功能模型迭代与设计构建迭代可以同步进行。
- 实现。将最终软件增量（一个可操作的原型）置于可操作环境。这个增量不见得100%完成了，即便是已经完成之后也可能需要改变，此时 DSDM 开发转向功能模型迭代继续进行。

---

① Stapleton J.. DSDM-Dynamic System Development Method: The Method in Practice. Addison-Wesley, 1997

② Jim Highsmith. 敏捷软件开发生态系统. 姚旺生，杨鹏等译. 北京：机械工业出版社，2004

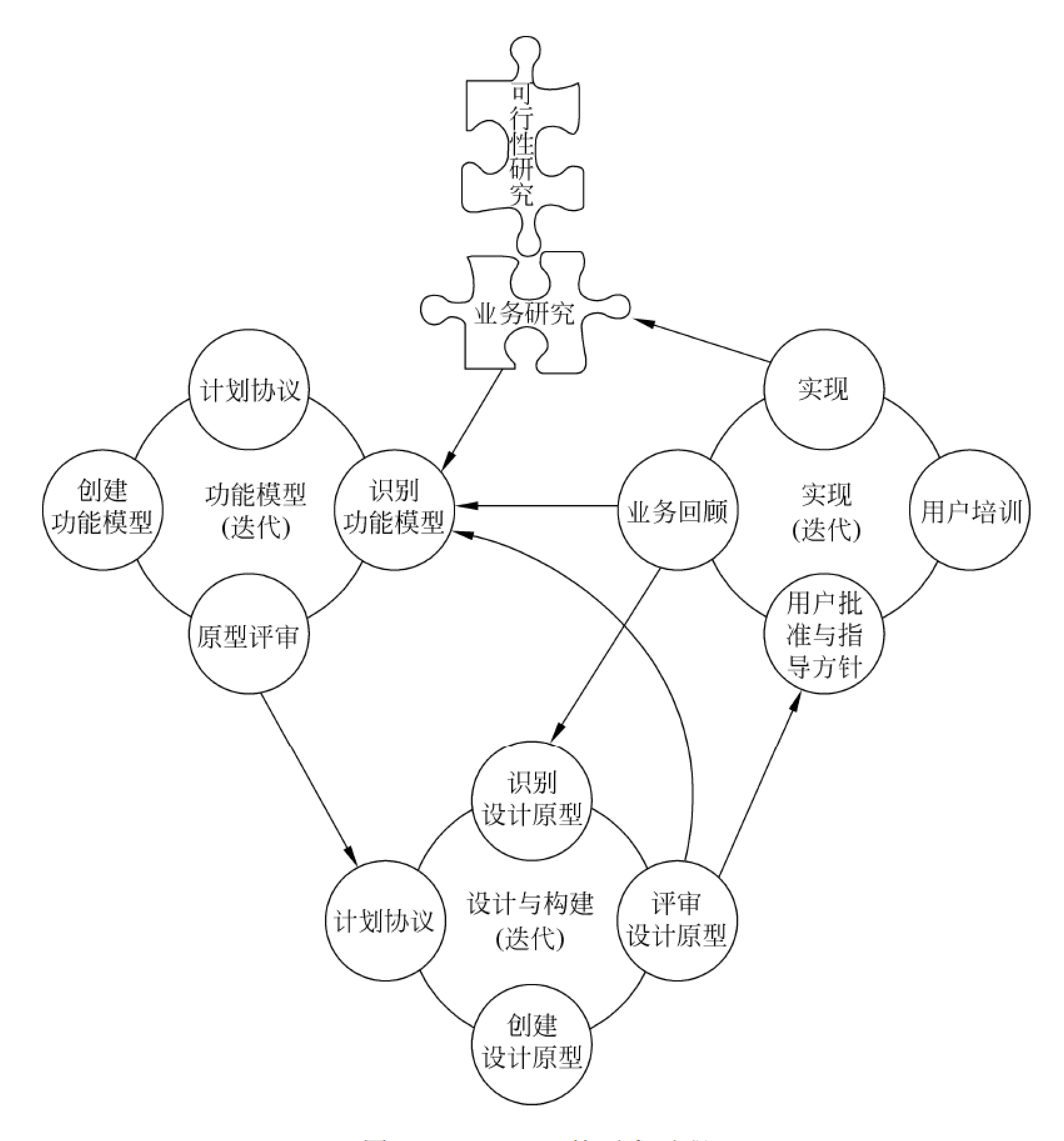


图 16-2 DSDM 的开发过程

在实际项目中,原型更像是那些将被改进为最终产品的代码,而不是传统的被抛弃的原型代码。

16.2.4 Scrum

众所周知,飞机在起飞前及航行过程中,飞行员从空中交通管制处获得指令,飞机受命起飞,并按照既定路线飞行。若情况发生变化,如气流颠簸,飞行员也是在获准后方可改变飞行高度。当飞机临近机场时,飞行员将获知着陆跑道和下机口的信息。然而,若是开车,人们便可以于任意时间选择所需的任何路线,当然在此过程中必须遵守交通法规。在车内,我们是独立的个体,在驾驶游戏规则的框架内,依照自身的最大利益做出决策。每天,有成千上万人驾驶汽车,没有中央控制和调度服务,仅在简单的交通法规框架内,便可完成目标,到达目的地。实际上,系统越复杂,中央控制系统崩溃的可能性越大,这便是公司分散权力,政府降低管制力度的原因。

Scrum 沿袭此思路,将控制权从中央计划及调度中心下放到实施工作的个体团队,将小型团队转化为自身命运的管理者。Scrum 团队接受挑战,寻找应对挑战的方法,发挥创意,避开工作障碍,这一切都是中央控制及调度系统无法预先安排的。如果团队规模合适,能激

发各成员的参与积极性，同时团队成员能意识到他们对自身命运的掌控，那么各成员的经验、意见和想法便可以得到充分利用；若团队成员信仰共同目标，便会设法实现它。而一旦整个团队达成互信，致力于向客户交付商业价值，且有权自主决定完成任务的方式，并拥有充足的资源，他们必定会成功。

Scrum 原则①和敏捷宣言一致：

- 组织小型团队以达到“沟通最大化，负担最小化，非语言描述、非形式化知识”。
- 过程对技术和业务变化必须具有适应性，以“保证制造具有最好可能的产品”。
- 过程生产频繁发布“可检查、可调整、可测试、可文档化、可构建”的软件增量。
- 开发工作和开发人员划分为“清晰的、低耦合的部分或包”。
- 坚持在产品构建过程中进行测试和文档化。
- Scrum 过程提供“在任何需要的情况下都能完成产品的能力”。

Scrum 建议由小型团队完成工作，既能够尽可能降低沟通负担，也有利于每天有效地召开 Scrum 会议。由 7 个人组成的小组是最合适的，既能够最大化能达到的功能，又能最小化沟通负担。Scrum 还建议每个小组至少有一位很有经验的工程师，以便指导小组其他成员。这种角色随时准备帮助小组的其他成员，有助于充分利用开发人员这种能力受限资源，通过迅速解决问题而避免出现窝工。

Scrum 的所有实践围绕着一个迭代、增量的过程骨架展开，如图 16-3 所示，下方循环代表开发活动的迭代，这种循环相继发生，每次迭代的产出成果便成为产品的增量；上方迭代代表迭代过程中的每日检查，团队成员举行会议相互检查工作，进行适当调整。需求列表是推动迭代的主要力量。

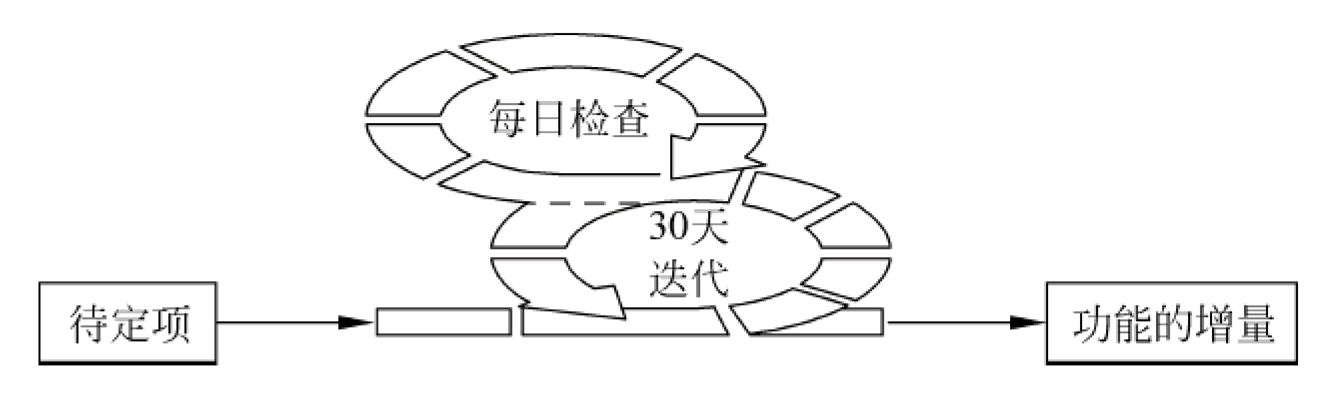


图 16-3　Scrum 过程流

该骨架运行方式如下：每一迭代初期，团队挑选出他们认为在该迭代结束时能转化为相应完整功能增量的部分；迭代其余时间内，团队不受干涉，努力工作；迭代结束时，团队展示完成的功能增量，请利益相关者进行检查，以对项目做出及时调整。Scrum 的核心在于迭代，团队首先浏览开发需求，考虑可用技术，并对自身技术及能力做出评估，然后共同确定构建功能的方案，并每日调整方法，以应对新的复杂问题、困难和出乎意料的情况，团队选择最佳方案去完成任务。所有工作在迭代周期内完成，每个迭代周期为 30 日。在每个迭代结束时要进行评审，可以向客户演示功能；也可以是团队总结上一个周期的经验教训，并用来在下一个周期中进行改进；还可以鼓励团队不断提高生产率。

Scrum 并非说明性的过程，它不描述在各类具体情况下的应对措施，仅提供一个框架及一组事件规则，确保可视性，而 Scrum 方法的实施者必须精确了解项目进展情况，及时做出

① http://www.controlchaos.com/

调整,保证项目顺利运行,达到预期目标。采用 Scrum 的实践者,应该充分利用常识,而不是坐待指令。

一家软件公司在使用 Scrum 之前,计划 12 个月内发布两项产品,然而,在成功使用 Scrum 后,5 个月内就开发好这两项待发布产品的大部分功能。

Scrum 带来的另一改变,可用建造房屋来加以说明。在全部房屋建设完成之前,买主不能搬入屋内居住。假设有一种增量、迭代的造房方式,依照它整栋住宅可以分房间逐步建成。第一个房间率先装好管道设备、电路和基础设施,而后随建筑完成情况延伸至各房间。一旦买主认为已有足够的房间建造完成,便可迁入,其余房间则可按买主的需要继续建造。Scrum 依照这一方式为客户开发软件。基础设施完成后,各项功能便分批交付给客户,在开发周期前期,客户便可使用部分系统了。由于能够提前体验系统情况,客户可以决定系统开发的次序及部件,并在相应部件完成后付诸使用。如果认为部分功能就能满足需求,客户甚至可以选择仅开发部分系统。

Scrum 能有效运作有两个原因:它极大地缩短了用户与开发者,预期目标与实施状况,投资与投资回报之间的反馈回路;它可就一个问题集思广益。

16.2.5 Crystal

Cockburn 将软件开发刻画为:“一种资源有限、合作完成的文明和交流活动,其首要目标是交付有用、可工作的软件,第二目标是为下一次行动做准备”[①]。

Crystal 是一个以人驱动的方法,其目的是发展一种提倡“机动性的”软件开发方法。可以用简短的话语做如下概括[②]:

> *总设计师和 2～7 名开发人员在一个大办公室或相邻办公室内,使用白板和挂图等信息传播器,方便联系到专家用户,干扰已排出,每一个或两个月(最长一个季度)把可运行、已测试以及有用的代码交付给用户,周期性地反思和调整工作惯例。*

要了解 Crystal 方法的生效方式,可提出两个特定问题:“团队在工作时以何为中心?”“我们能否将项目带入一个更加安全的区域?”执行 Crystal 方法应该将重点放在为项目开发而准备的七大关键特征上:

(1) 经常交付。
(2) 反思改进。
(3) 渗透式交流。
(4) 个人安全。
(5) 焦点。
(6) 与专家用户建立方便的联系。
(7) 配有自动测试、配置管理并支持经常集成的技术环境。

Crystal 方法强调人、交互、团队、技能、才智和交流。每个人或团队都有其独特的才智和技能,每个团队都应该为他们量身订制过程,但是,让每个团队从起点开始开发独特的过

① Cockburn A. What Is Agile and What Does It Imply? http://crystalmethodologies.org/
② Alistair Cockburn. 小团队的敏捷开发方法. 马振晗,罗海花译. 北京:清华大学出版社,2006

程是低效的。水晶是有多面的，每个不同的面都在一个基本核心上，在此，基本核心代表价值和原理，每个面代表特殊的元素：技术、角色、工具和标准。

在 Crystal 方法中，启动项目时可供选择的策略有以下几个。

(1) 360 度全方位考察——项目承包活动的一部分。在启动项目时，团队必须从不同方面对项目进行考察，根据考察结果决定是否继续项目的开展。可以用取样的方法进行。

(2) 早期胜利——一个项目管理的策略。胜利可以使团队更加团结，并让成员更加自信，一个小小的成功就能使团队增加实力与自信。

(3) 灵活程序框架——一个体系机构/项目管理的策略。一个灵活的程序框架是指运行小型端对端功能的一小部分系统，无需利用系统的最终架构，但是应该把架构的主要部分连接起来，这样架构以及系统功能就能同时改进。灵活程序框架并非一个完整的、功能强大的程序，只是一些框架而已，并没有实质的应用功能内容，但是随着每次增量的交付，它们会日趋完善，也会有越来越多的功能被添加到框架中。

(4) 增量重建——一个与灵活框架相关的策略。系统架构需要经常通过更新灵活程序框架进行改进，团队分阶段地对架构进行改进，同时要保证系统的正常运行。团队可以利用增量开发对系统的基础结构、系统架构以及对系统的最终功能进行修改。采用此策略，可以带来如下好处。

- 有需要时，能够更轻松地修改架构。
- 功能开发小组和基础结构小组可以同时工作。
- 最终用户可以提前对系统推荐的功能进行审查，而后提供建议，使其更加符合业务用途。
- 运行系统能暴露出一些在预想实验中无法发现的缺陷。

(5) 信息传播器——一个交流的策略。信息传播器是一种信息公告，通常放置在成员工作和路过时能看得到的地方，成员无需向他人询问便可获得他们所关切的信息，这意味着更多的交流、更少的干扰。一个好的信息传播器应该大而明显、明晰易懂、定期进行更换、容易及时进行更新。信息传播器一般用于传播状况信息，如当前迭代工作、当前任务、已编写完成的测试个数、已交付的用例的个数、上次反思研讨会的结果等。

360 度全方位考察给了团队一个发现致命错误的机会，不需要花费太长的时间就可以获得很高的回报价值。早期胜利、灵活程序框架和增量重建这些策略要互相配合。信息传播器在项目开发的各个阶段都非常有用。这些策略能够给予项目好的开端。

16.2.6 特征驱动开发

特征驱动开发（Feature Driven Development，FDD）本质上是一种软件管理方法，而不是软件开发方法。FDD 没有引入新的软件开发方法，而是利用业界精英通过管理手段创新总结出的最佳实践。FDD 过程反映了从面向过程方法中学习的过程，以客户为中心、以结构为中心、注重实效。有些敏捷方法论学者把 FDD 描述为“不够敏捷”，可能是因为 FDD 看起来更像是成熟的软件开发方法。FDD 包括策划和前期建模与设计，由以下 5 个阶段构成[①]。

---

① David J. Anderson. 软件工程的敏捷管理. 韩柯，等译. 北京：机械工业出版社，2004

1. 开发整体模型

FDD 把建模看做是需求发现过程的一部分,并吸收市场开发团队参与。建模工作产生 UML 类图,对经过不同团队相互沟通得到的业务行为建模。这个建模步骤还包括一种同时进行的研究非特性需求和建立待开发系统体系结构模型的活动。

2. 特性列表

FDD 的基本要素是特性,特性是一小块增值内容:一种可以在两周或更短时间内交付的实在结果;一种频繁产生的结果,具有一定的质量测度;一种能够投入使用的结果。

使用特性有如下好处。

(1) 特性是小块可发布功能,用户可以方便地描述、轻松地理解其相互关系,更好地评审以发现歧义性错误和遗漏。

(2) 特性可以组织为具有层次关系的业务相关分组。

(3) 特性很小,其设计、代码都可以很容易、很有效地检查。

(4) 项目计划、进度和跟踪都由特性层次驱动,而不是可任意调整的软件工程任务集。

FDD 定义了 4 层体系结构:UI——用户界面、PD——问题域(业务逻辑)、DM——数据管理、SI——系统接口,其中每一层都可以开发详细模型。特性列表采用一种能够定义很小块开发工作的语言规则列出。每类特性都采用可重复的规则定义。

PD 特性可采用以下结构编写:

<行动><结果>[by|of|from|for|to]a[n]<对象>

例如,一个电子商务应用的特性可能如下所示:

```
Add the product to a shopping cart.
Display the technical - specifications of a product.
Store the shipping - information for a customer.
```

一个特性集将相关特性分在一个业务相关的类别中,定义如下:

<行动> a[n]<对象>

例如,Making a product sale 是一个特性集,包含上面提到的及其他可能的特性。

UI 特性的规则是不同的,它采用一种状态图建模,如图 16-4 所示的状态图,是表示层代码的精确蓝图,表示层代码采用Ⅱ型"模型-视图-控制器"的变种模式实现。状态图的一些状态被设计为视图,状态图的状态转移采用引发状态转换的事件名称标出。

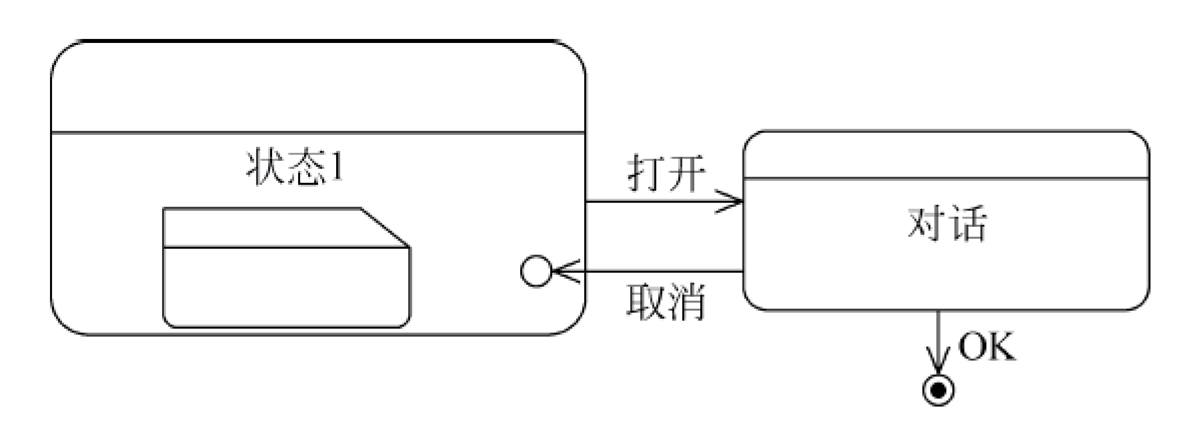


图 16-4 状态图 UI 模型举例

UI 特性可以是以下两种特性之一：视图类，例如“搜索结果表”；控制器类，例如“搜索”事件。

SI 和 DM 的规则与 PD 规则类似，但是结果和对象用词是与系统有关的，而不是与领域有关。

3. 按主题域策划

通过把特性列表上的特性分成特性相关的叫“特性集”(FS)和“主题域”(SA)的集合，对项目进行策划，然后团队再与项目出资方就交付主题域和特性集的交付进度计划达成一致。

4. 按特性集设计

问题域(PD)特性集叫做“业务活动”，集合中的特性按逻辑分组，把这些特性分批次进行设计。特性集应该是潜在可交付客户的工作块，是可能作为一个组件交付的一组一致的客户增值特性，是一组相关业务特性。特性列表必须排序，以便缓冲特性集。

FDD 的设计有详细、深入的 UML 建模，包括增强类图，开发活动中每个特性的 UML 时序图。

5. 按特性进行构建

包括打包第四步特性集中的较小的特性批，开发这些特性的代码，进行单元测试。这种特性批叫做“首席程序员工作包”(CPWP)，其规模应该适合一个特性实现团队在两周之内完成，一般由 5～10 个特性组成。领导特性实现的首席程序员负责与项目经理或项目秘书以及项目组其他程序员一起协商如何选择 CPWP 中的特性。

FDD 最适合作为一种需求过程的积极活动，通过由企业拥有者和开发人员组成的联合小组完成建模的情况。这样能够开发出详细模型，并进行准确策划。对于质量低或模糊的高层需求，编写特性列表很困难，将特性映射到模型上的类会很困难，此时，最好使特性看起来尽可能更像是模板，并不过多考虑与模型的关联。有时特性的粒度很粗，有时特性看上去更像是任务，而不是一小块客户增长特性。

16.2.7 精益开发

精益(Lean)思想诞生于 20 世纪 40 年代末，当时，由于缺乏足够的资金，丰田公司制定了一个新的生产系统，核心是追求零库存、零浪费、零不良、零故障、零灾害、零停滞的较为完美的生产系统。1/3 的时间、1/3 的预算、1/3 的缺陷率，这就是在软件开发中精益开发方法的目标。

Poppendieck 认为精益思想遵循以下七大原则①。

1. 消除浪费

消除浪费是最为基本的精益原则，是所有其他原则必须遵守的首要原则。因此，实施精

① Mary Poppendieck，Tom Poppendieck. 敏捷软件开发工具——精益开发方法. 朱崇高译. 北京：清华大学出版社，2004

益开发的第一步是学会识别浪费；第二步是揭示最大的浪费源，并消除它们。精益开发思想的策划者，被称为丰田生产之父的 Taiichi Ohno 认为，任何不能为顾客创造价值的事物都是一种浪费。丰田系统的策划者之一 Shigeo 确认了制造业的 7 种浪费类型，曾帮助众多制造部门经理识别出自己从未想到过的浪费，在软件开发中也有相应的 7 种浪费，分别如表 16-1 所示。

表 16-1 7 种浪费

制造业中的 7 种浪费	软件开发中的 7 种浪费
库存	部分完成的工作
额外过程	额外过程
生产过剩	额外特性
运输	任务调换
等待	等待
移动	移动
缺陷	缺陷

库存会隐藏质量问题，使产品逐渐过时，还会阻塞销售渠道；而部分完成的软件开发工作存在失效的趋势，会妨碍其他可能需要进行的开发工作，会占用尚未产生成果的投资资源。部分完成的软件开发工作可能会带来巨大的财务风险。

2. 增强学习

开发是一个发现过程，而生产则试图减少变化，精益开发方法会产生完全不同于生产的实践。人们试图制定标准化的过程，以减少变化，每次获得可重复的结果。但开发工作并非旨在产生可重复的结果，而是针对不同客户问题提供适当的解决方案，最好的方法就是增强学习。

3. 尽量推迟决策

推迟决策的开发实践在设计不确定性的领域中行之有效，因为它们能提供基于选择的方法。在不断演变的市场中，保留可选设计方案要比早期做出承诺更有价值。在开发复杂系统时，推迟承诺的关键性策略是将可更改性嵌入系统。

4. 尽快交付

快速开发具有很多优点：没有速度就不可能推迟决策，也没有可靠的反馈；在开发过程中，周期越短，能学到的东西就越多；速度能确保客户得到当前所需的东西，而不是过时的东西；速度还允许客户推迟决定他们真正需要的东西，直到他们了解了更多的情况。尽量压缩价值流是消除浪费的基本精益策略。

5. 授权团队

一流的执行过程取决于对细节的修正，没有人比实际承担工作的人员对细节更为了解，

让开发者涉足决策细节至关重要。人们需要的不仅是一系列任务，更重要的是要有内在动机，只有有来自对创造性劳动的荣誉感，才能使人干劲十足。首先要制定明确、令人信服的目标，确保目标可以实现，还要让团队与客户接触，让团队自己做出承诺，并消除团队的疑惑。要让团队成员能独立开展工作，为他们提供发挥才智所需要的空间，让他们在工作中有自己的想法，要鼓励而不是挑剔。

6. 嵌入完整性

产品完整性包括外部和内部完整性，在这里分别将其称为感知完整性和概念完整性。感知完整性表示整个产品达到功能、可用性、可靠性和经济性的平衡，使客户感到满意。概念完整性表示系统的核心概念能作为一个稳定的内聚整体共同发挥作用。除此之外，软件还需要具备额外的完整性水平——必须在一段时间内维持其有用性，因此软件通常需要进行适当的演进，以适应将来的需求。拥有完整性的软件具备一个内聚构架，具有很高的可用性和适用性，并能得到维护、调整和扩展。

7. 着眼整体

系统是由相互依赖和交互的各部分组成，为某个目的联系在一起的。系统实现其目的的能力取决于各部分的协作水平，而不仅仅是各部分性能的高低。而在产品开发过程中，各个领域的专家都倾向于尽量夸大产品某一部分的性能，而不是凝聚系统的整体性能，这样共同利益就会受到损害。最好的组成部分不一定能构成最好的系统，这并不是简单的 1+1=2。因此，要大处着眼，小处着手。

Bob Charette 则认为精益开发的关键要素是：①迅速创建可见的用户价值；②构建的软件适应变化；③仅创建必要的功能；④挑战性、顽固性和信任在精益开发的延伸目标中结合[①]。Jim Highsmith 从这 4 个要素扩展开来，得到了以下 12 条原则。

(1) 满足客户是最高宗旨。
(2) 坚持最高性价比原则。
(3) 成功依赖积极的客户参与。
(4) 每个 LD 项目都需要一个团队的努力。
(5) 事事可变。
(6) 领域的解决方案，而不是点的解决方法。
(7) 完全，但不是全面构建。
(8) 宁愿今天解决 80%，也不等到明天解决 100%。
(9) 最小化是根本。
(10) 需求决定技术。
(11) 产品增长是特性增长，而不是规模增长。
(12) 理解 LD 所能解决的问题范畴，不要把它用于其能力之外。

---

① Jim Highsmith. 敏捷软件开发生态系统. 姚旺生，杨鹏等译. 北京：机械工业出版社，2004

16.3 设计自己的敏捷方法

敏捷是动态的,适应于具体情况的、迎合变化的和自我完善的。敏捷方法不仅能够应对变更,而且特别善于应对某一种特定类型的变更,即突击要求。不同的方法应对突击要求能力也有所不同,不是任意选择一种敏捷方法就能适应具体的项目实施。极限编程最适合几乎没有稳定性的领域;Scrum 排除了在当前迭代中出现突击要求的可能性,突击要求可以推延到下一轮;FDD 中有变更控制过程,可以接受变更,但可能有很多正在处理的过程必须停下来为突击要求让路,使成本偏高。表 16-2 给出了每个敏捷方法的敏捷性。

表 16-2 敏 捷 等 级

	CMM	ASD	Crystal	DSDM	FDD	Lean	Scrum	XP
混沌有序组织	1	5	4	3	3	4	5	5
协作价值	2	5	5	4	4	4	5	5
简单	1	4	4	3	5	3	5	5
响应能力与适应性	2	5	5	3	3	4	4	3
技术优势	4	3	3	4	4	4	3	4
协作实践	2	5	5	4	3	3	4	5
敏捷等级	1.7	4.8	4.5	3.6	3.6	3.9	4.7	4.8

说明:数字代表敏捷程度(数值越大越敏捷)。

敏捷方法的设计和调整应该基于以下几点。

(1) 对方法的现实期望值。

(2) 对各个方法元素的明确定义,包括实践系统。

(3) 一组有效的设计原则。

(4) 一个全面的框架、项目模板和开发说明。

(5) 一套设计步骤。

(6) 用于定制和剪裁方法的过程。

(7) 扩展到大型项目的指导方针。

未来的业务环境会继续保持混乱,要使过程变得精简或使文档规模尽量减少,应使它们更敏捷。"反映现实世界"是敏捷软件开发的实质,反映了实用的被实践过的,而不是理论化的和强加的。敏捷开发解决了如何在高度易变、需求迫切的情况下更快地开发出更好的软件。银弹是不存在的,敏捷开发的根本在于相信在混乱的业务环境下有不可预测性,相信人们在面对这种不可预测性时能成功交付软件的能力的可预测性。

本 章 小 结

软件工程的敏捷理念强调 4 个关键问题:具有控制力的自我组织团队对所开展工作的重要性;团队成员之间、开发参与者与客户之间的交流与合作;对变更的认识;以及强调快速交付软件让用户满意。

极限编程XP是应用最广泛的敏捷过程之一,按策划、设计、编码和测试4个框架活动组织,建议重构、结对编程、测试驱动开发等新颖有力的技术,保证了团队能创建体现用户指定优先级特征和功能的频繁的软件发布。

自适应软件开发ASD强调人的合作和团队的组织,按照预测、协作和学习3个框架活动组织,使用迭代过程,其迭代是目标驱动的、基于部件的、风险驱动和容变的。

动态系统开发方法DSDM,通过在可控项目环境中使用增量原型开发模式完全满足对时间有约束的系统的构建和维护,定义了3种不同的迭代:功能模型迭代、设计与构建迭代以及实现迭代,在3个迭代前还增加了可行性研究和业务研究这两个生命周期。主要倡导时间调度的使用。

Scrum强调一系列软件过程模式的使用,每一个过程模式定义一系列开发任务并允许团队以使用其项目的方式构建过程。Scrum建议小型团队工作,所有的实践围绕一个迭代增量的过程框架展开。

Crystal是一种以人驱动的方法,提倡“机动性”,是一系列敏捷过程模型,强调人、交互、团队、技能、才智和交流。

特征驱动FDD本质上是一种软件管理方法,而不是软件开发方法,它并没有引入新的软件开发方法。FDD反映了从面向过程方法中学习的过程,以客户为中心,以结构为中心,注重实效。它包括策划和前期建模与设计。

精益开发Lean的核心是追求零库存、零浪费、零不良、零故障、零灾害、零停滞,目标是1/3的时间,1/3的预算,1/3的缺陷率。其指导原则是:消除浪费、增强学习、尽量推迟决策、尽快交付、授权团队、嵌入完整性、着眼整体。

“反映现实世界”是敏捷软件开发的实质,敏捷开发解决了如何在高度易变、需求迫切的情况下更快地开发出更好的软件。不同的敏捷方法应对突击要求能力也有所不同,不是任意选择一种敏捷方法就能适应具体的项目实施。模型的复杂度、类型和规模必须根据所构建的软件来调节,通过提出一系列核心和补充建模原则,敏捷建模给实践者的分析、设计提供了有用的指导。

思考与练习

1. 用自己的语言描述敏捷性。
2. 讨论本章所介绍的各种敏捷方法的特点,说明它们各自适用于什么情况。
3. 为什么说迭代过程更容易应对变更?本章所讨论的敏捷过程是否都是迭代的?
4. 讨论本章所介绍的敏捷方法是否符合敏捷性的12条原则?
5. 查找资料,除了本章所讨论的敏捷方法以外,选择一种敏捷方法进行介绍。
6. 目前最热门的软件应用领域之一是基于Web的系统和应用,能否把敏捷方法应用于其中?

第17章 Web工程

Web 开发还处于青年时期……正如大多数的青年人一样,在他试图离开他的父母时,他想要像成年人那样被接受。如果 Web 开发要完全成长起来,就必须向更老练的软件开发世界学习。

——Doug Wallace

随着网络技术和互联网的发展,Web 已成为世界范围内最大和最有效的信息获取和发布的媒体。只需要一个 Web 浏览器,就能实现一般应用程序所能实现的功能,而且只要能上网的地方,就能远程地实现信息的输入和处理,这是一般应用程序所不具备的特点,也是 Web 应用程序的优势所在。那么如何建造一个高质量的 Web 应用程序呢?本章的内容将简要介绍开发高质量 Web 应用的原理、概念和方法。

17.1 基于 Web 的系统及应用

17.1.1 Web 应用的发展

Web 应用是从 Web 站点发展而来,第 1 批 Web 站点是 Tim Berners-Lee 在欧洲粒子物理实验室时建立的,它们形成了一个分布式的超媒体系统,使研究者们能够直接从同事们的计算机上访问他们公布的文档和信息。文档是通过浏览器来访问和浏览的。浏览器是一个运行在客户计算机上的软件应用程序。通过浏览器,用户能从网络上的其他计算机上对文档发出请求,然后把那些文档交付到用户的显示器上。为了浏览一个文档,用户需要启动浏览器,然后输入文档名和文档所在的主机的名字。浏览器向主机发送一个对文档的请求。请求被一个称为 Web 服务器的软件应用程序所处理。Web 服务器是一个应用程序,它通常作为一个服务而运行。它在一个特定的端口上监控网络活动,通常是端口 80。浏览器通过该网络端口向 Web 服务器发送一个针对一个文档的特殊格式的请求。Web 服务器接受该请求,在本地文件系统中找到该文档,然后把它传送给浏览器,如图 17-1 所示①。

---

① Jim Comallen. 用 UML 构建 Web 应用. 第 2 版. 陈起,英宇译. 北京:中国电力出版社,2003

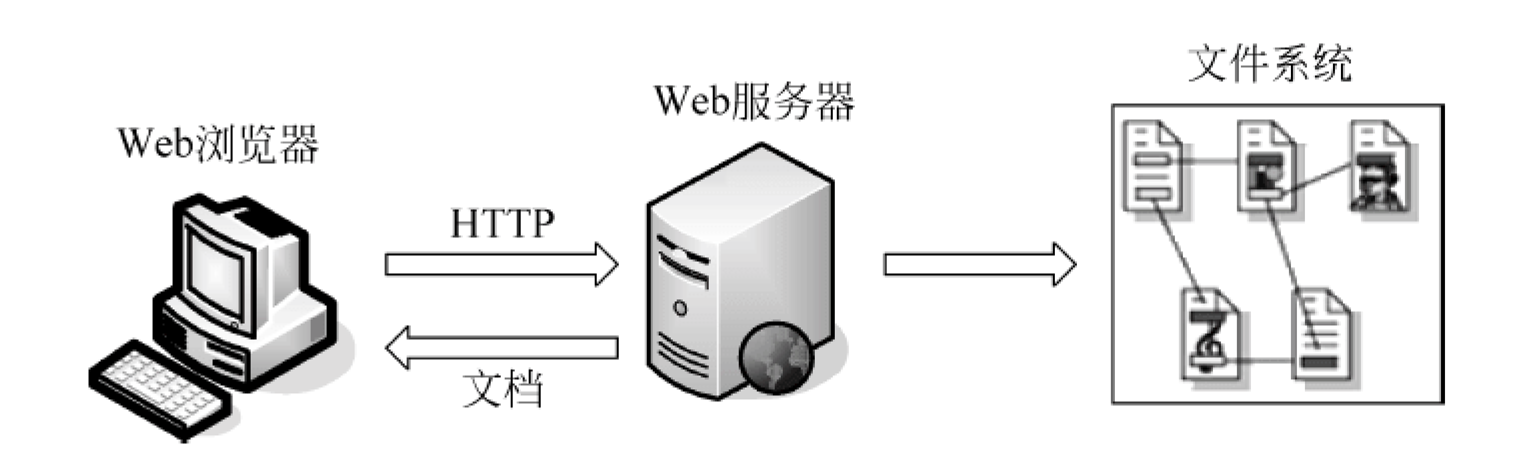


图 17-1 Web 站点架构

Web 站点主要由大量相对静态的页面和通用网关接口(Common Gate Interface,CGI)组成,设计和建立比较简单,交互性较弱,个性化不强。

随着网络技术和 Internet 的迅速发展,人们越来越多地接触到不同类型的基于 Web 的应用系统。从初始的信息型到现在的 Web 服务,复杂程度也越来越高,如图 17-2 所示。

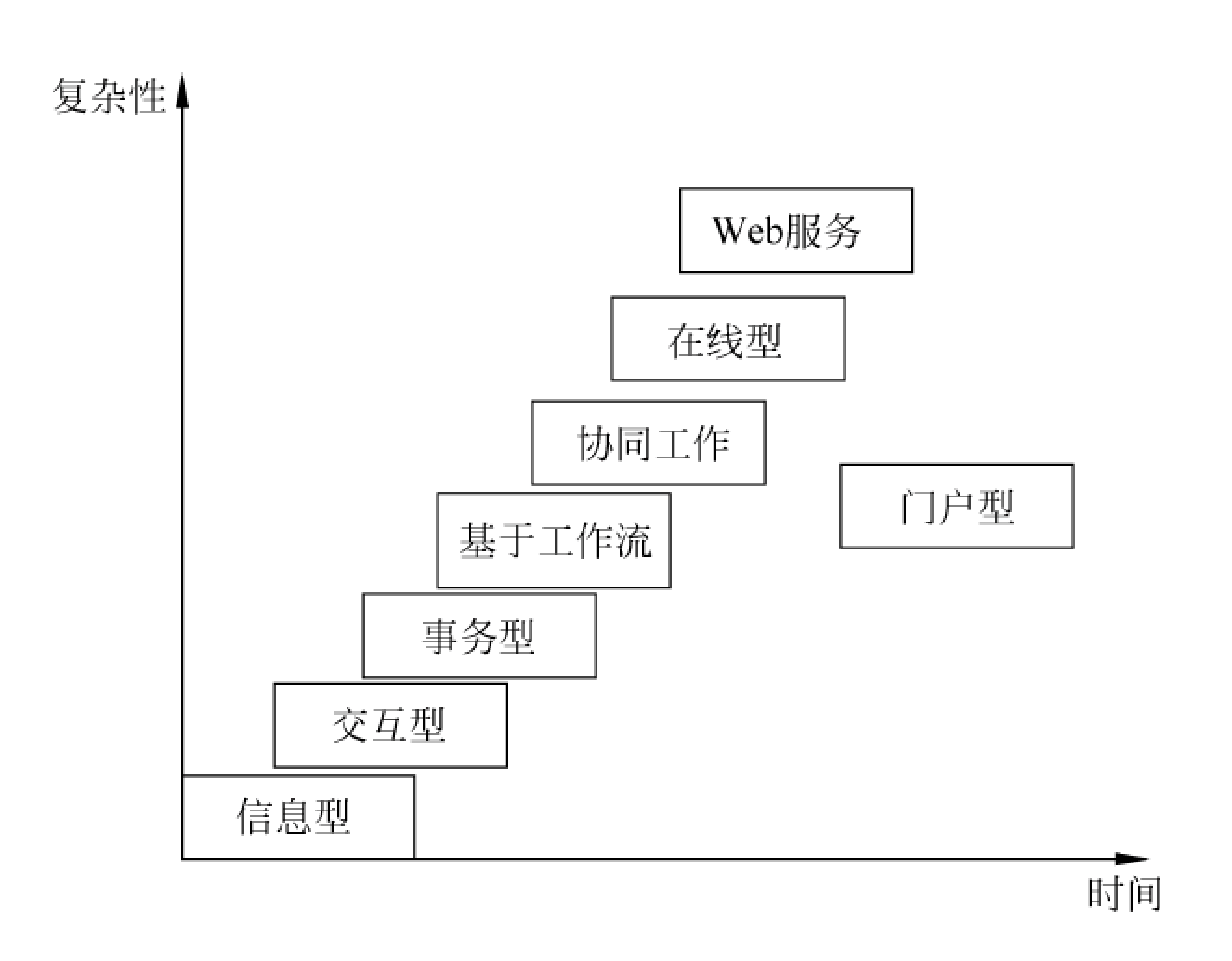


图 17-2 Web 应用的发展[①]

Web 应用建立在 Web 站点基础之上,通过添加业务功能的方式加以扩展,Web 应用与 Web 站点的区别在于用户是否有影响服务器上业务逻辑状态的能力。从本质上讲,Web 应用利用 Web 站点作为一个业务应用的前端。所以说只有这样的系统才可以认为是一个 Web 应用:在系统中是以浏览器作为客户与应用交互的界面来执行业务逻辑。一次交互可以分解为以下 3 步[②]。

① Damiamo Distante, Paola Pedone, et al.. Model-Driven Development of Web Applications With UWA, MVC and Java Server: Faces

② Pierre-Alain Muller . Philippe Studer, et al.. Platform independent Web application modeling and development with Netsilon, Software & System Modeling

第 1 步请求——用户通过已经在 Web 浏览器上可视的 Web 页面向 Web 服务器发送一个请求，请求能以表格或链接的方式发送给服务器。

第 2 步处理——Web 服务器接收到请求后，执行各种动作来说明一个包含请求结果的页面。然后这个页面被传递给发送请求的 Web 浏览器。

第 3 步应答——浏览器在合适的位置或者另一个浏览器窗口上显示请求的结果。

Web 应用架构如图 17-3 所示。

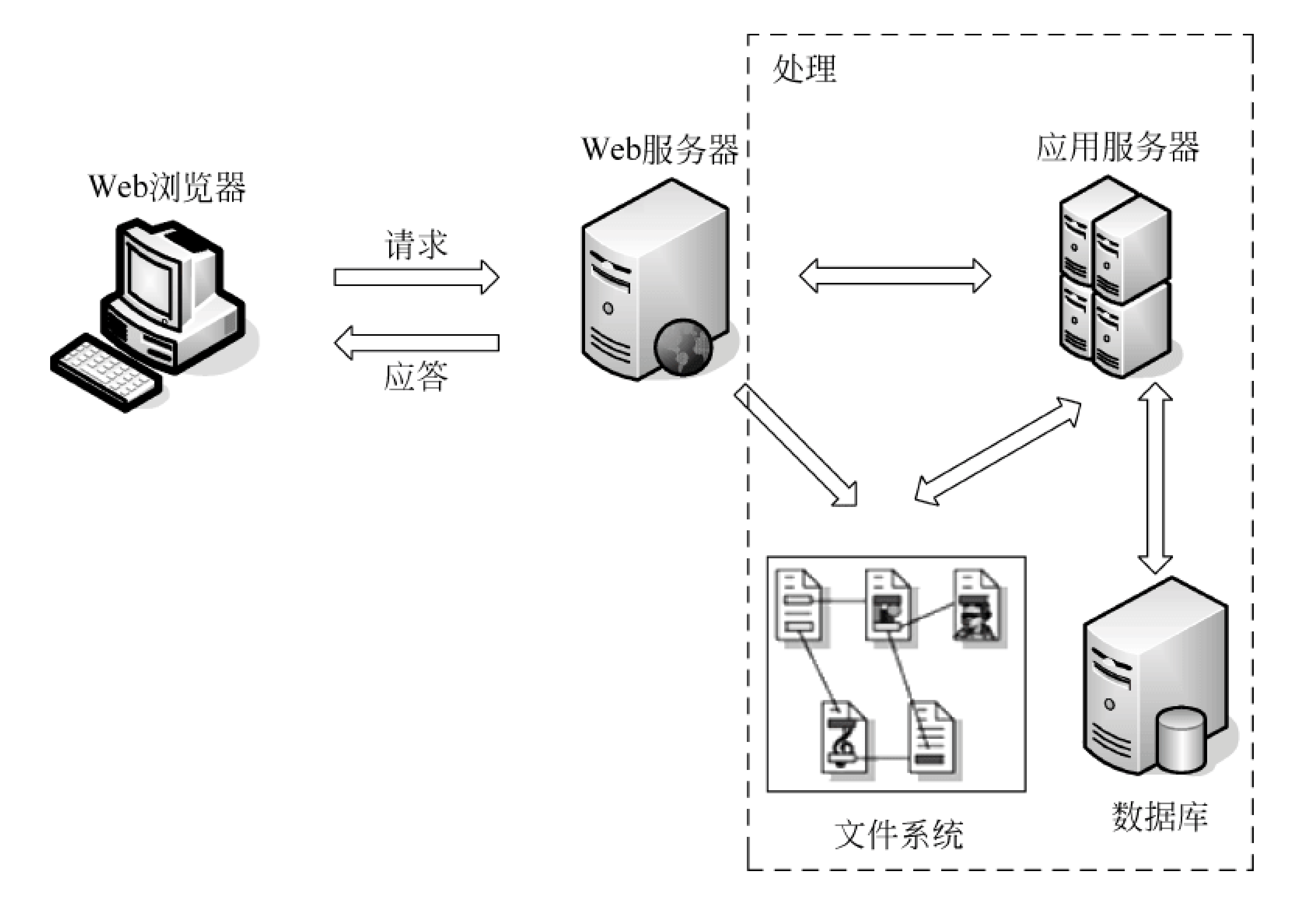


图 17-3 Web 应用架构

(1) Web 浏览器：其功能是为用户提供一个浏览文档的窗口，用户通过它向 Web 服务器发送请求。

(2) Web 服务器：主要响应浏览器端的请求，在传统的 Web 站点中，当 Web 服务器接到浏览器端的请求后，将响应并且返回相应的页面。在 Web 应用中，Web 服务器一方面实现上面的功能，另一方面它将根据请求与应用服务器发生信息交互。

(3) 应用服务器：因为浏览器与服务器之间遵循的是超文本传输协议(Hypertext Transfer Protocol，HTTP)，一旦一次响应结束，两者的连接就断开了，所以传统的 Web 站点难以进行复杂的数据和业务处理。应用服务器主要实现了对数据的处理和对数据库、文件系统的访问与操作。

随着互联网的广泛应用和延伸，以及企业信息计算的应用层次的不断深化，与之相对应的 Web 应用系统的体系结构也变得日趋复杂。Web 应用系统的体系结构从以前的 2 层发展到现在的 3 层或多层。典型的 3 层结构为表示层、业务逻辑层和数据存储层，如图 17-4 所示。

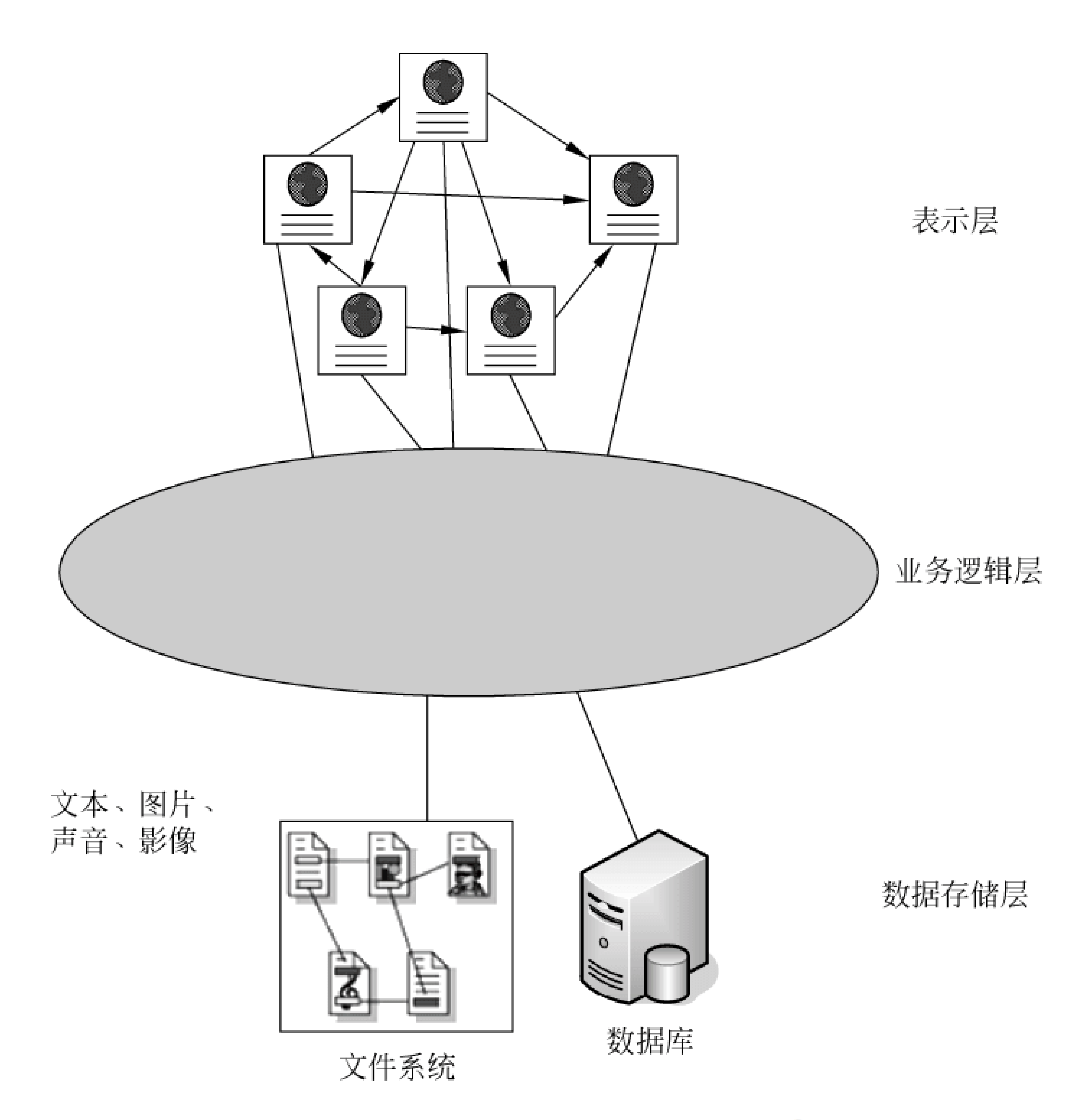


图 17-4 Web 应用的 3 层体系结构①

17.1.2 Web 应用的特点和分类

早期的 Web 站点仅包含链接在一起的超文本文件，这些文件使用文本和有限的图标来表示信息。随着时间的推移，Web 应用已经逐渐发展成为成熟的计算工具，这些工具不仅可以为最终用户提供独立的功能，而且已经同公司数据库和业务应用集成在一起。Web 应用不同于传统的计算机软件，在绝大多数 Web 应用中都有以下属性②。

(1) 网络密集性：Web 应用驻留在网络上，服务于不同客户群体的需求。Web 应用可以放置在某内联网(实现组织范围内的通信)或某外联网(网际间通信)上。

(2) 并发性：在同一时间可能有大量用户使用 Web 应用，而且在很多情况下，最终用户的使用模式存在很大差异。

(3) 无法预计的负载量：Web 应用的用户数量每天都可能会有数量级的变化。周一显示有 100 个用户使用这个系统，周二就可能会有 10 000 个用户。

(4) 性能：如果一位 Web 应用用户的请求必须等待很长时间，该用户可能会放弃并转向其他地方。

(5) 可得性：尽管期望 100％的可得性是不切实际的，但是大多数的 Web 应用用户通

① Gennaro Costagliola，Filomena Ferrucci，Rita Francese web engineering-models and methodologies for the design of hypermedia applications. Handbook of Software Engineering and Knowledge Engineering

② Roger S. Pressman. 软件工程——实践者的研究方法. 郑人杰，马素霞，白晓颖等译. 北京：机械工业出版社，2007

常要求全天候的可访问性。

(6) 数据驱动：许多Web应用的主要功能是使用超媒体向最终用户提供文本、图片、音频及视频内容。除此以外，Web应用一般用来访问那些存储在数据库中的信息，这些数据库最初并不是基于Web环境的一部分，例如电子商务或金融应用。

(7) 内容敏感性：内容的质量和艺术性仍然在很大程度上决定了Web应用的质量。

(8) 持续演化：传统的应用软件是按一系列规划好的时间间隔发布进行演化的，而Web应用则会持续地演化。对某些Web应用（特别是Web应用的内容）而言，以分钟为单位进行更新，或者对每个请求进行独立运算是可能的。

(9) 即时性：尽管即时性（也就是将软件尽快推向市场的迫切需要）是很多应用领域的特点，将Web应用投入市场可能只是几天或几周的事情。Web工程师们必须使用经过修改的策划、分析、设计、实现和测试的方法以满足Web应用开发所要求的紧迫的时间进度安排。

(10) 保密性：由于Web应用是通过网络访问来使用的，要限制访问的终端用户的数量非常困难。为了保护敏感的内容，并提供保密的数据传输模式，只有在支持Web应用的所有基础设施和应用本身内部实现较强的保密措施。

(11) 美学性：Web应用具有吸引力的一个不可否认的部分是其观感。当要面向市场推销产品或想法时，与技术设计相比，美学可能同样事关该应用的成败。

这些属性是一般Web应用具有的，但其影响程度会有所不同。下面的应用类别是国内Web应用开发中最常碰到的[①]。

(1) 以页面为中心的Web应用系统。此类系统的服务器端主要由静态的页面和它们之间的超链接构成。用户通过浏览器向Web服务器发出一个请求，Web服务器接受浏览器的请求，向浏览器发送页面文件。

(2) 以数据库为中心的Web应用系统。此类系统的服务器端主要由嵌入了程序的页面和相关的数据库构成。用户通过浏览器向Web服务器提交服务请求，Web服务器分析用户输入的数据，执行相应的程序（通常要访问数据库），根据不同的数据内容将相应的执行结果（通常是数据库查询的结果集）以超文本标记语言（Hypertext Markup Language，HTML）的格式传送给浏览器。

(3) 以应用逻辑为中心的Web应用系统。此类系统的服务器端由页面、单独的应用服务器软件和数据库构成。由于此类系统的应用逻辑比较复杂，所以通常由单独的服务器程序来完成相应的应用逻辑。用户通过浏览器提交一些复杂的服务请求（如网上交易），Web服务器对请求格式转换后将其转发给应用服务器，应用服务器通过执行相应的应用程序来处理请求，并将结果返回给Web服务器，Web服务器根据结果生成页面，并将页面传给浏览器。

17.1.3 Web应用的开发团队

构建Web程序需要有一个具有不同技能、知识和能力的人组成的团队，可以把参加开发Web应用系统的人员分为Web项目管理人员、内容提供人员、Web开发人员、Web测试

① 张友生. Web工程实践研究[J]. 小型微型计算机系统，2004，25(9)：1607～1611

人员、Web维护人员、Web管理员、最终用户等[①]，如图17-5所示。在每一个类别中，根据具体技能或从事工种不同，又可定义不同的子类别(如Web开发人员又可分为系统分析员、程序设计员、美工人员等)。一个人可以属于不同的类别(如最终用户也可以是内容提供人员和Web发布人员)。



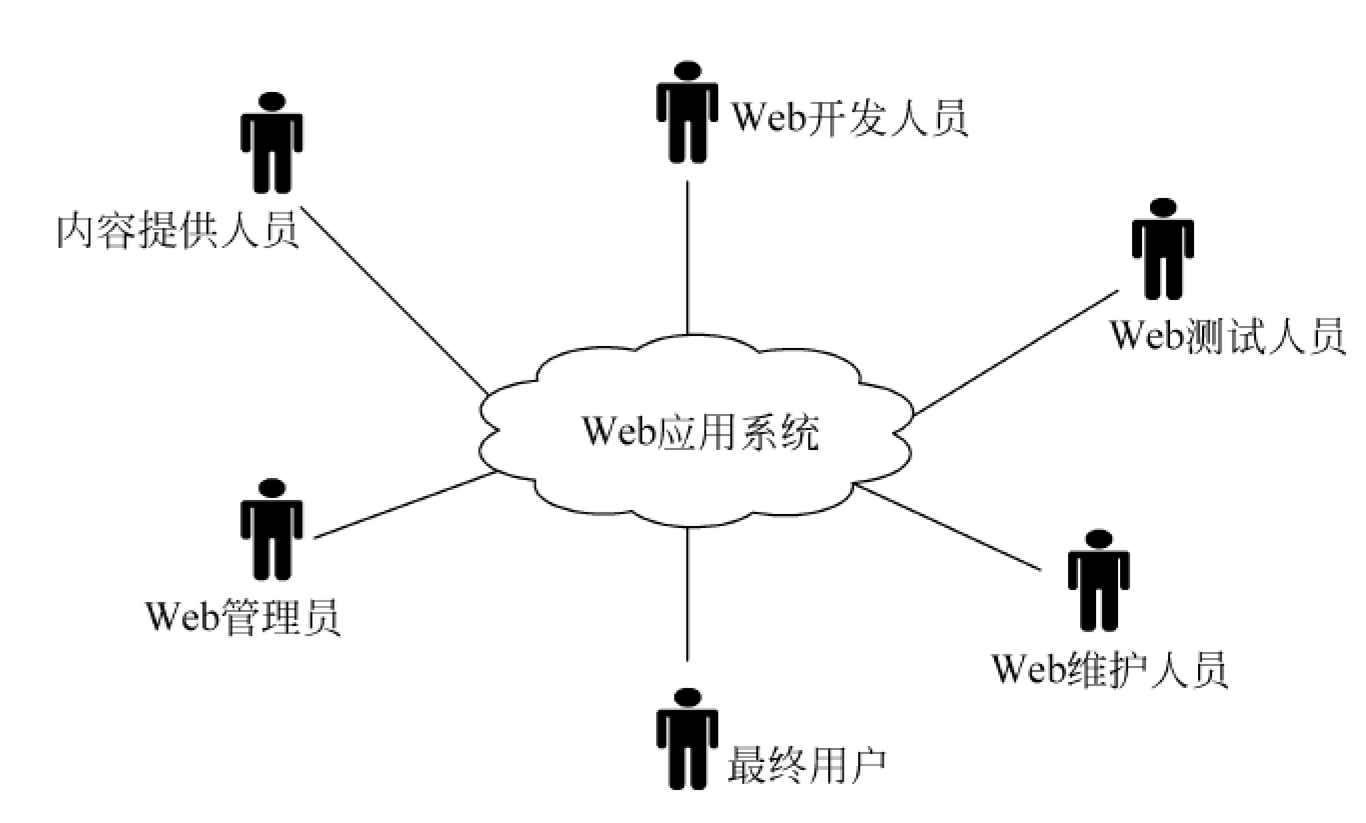


图17-5 Web开发团队结构图

(1) Web项目管理人员：一般指开发组织的领导层人物，能够对是否开发Web应用系统起决策和管理作用。

(2) 内容提供人员：可以是开发组织内部人员，也可以是外部人员或最终用户。

(3) Web开发人员：Web开发人员包括系统设计人员、程序设计人员、界面设计人员等专业技术人员。

(4) Web测试人员：对Web应用系统进行测试。

(5) Web维护人员：经过适当培训的Web维护人员能从事一些技术支持工作，其作用包括更新、维护Web应用系统。

(6) 最终用户：最终用户是Web应用系统的客户，他们使用带有浏览软件的计算机(终端)，代表Web应用系统的入门级参与者。虽然处于基本层次的最终用户只需要会操作带有软件的计算机，但也需要学习一些如导航技能、使用复杂的搜索引擎的技能及传送文件的技能等高级的操作。

(7) Web管理员：Web管理员的作用是进行Web网络管理，涉及Web软硬件操作的技术技能、网络和通信技能。技术方面包括记录文件版本、数据库操作、安全和存取权限、通过CGI程序或类似的扩展程序进行服务端操作。另外Web管理员还需具备与Web性能相关的知识。

---

① 张有生. Web工程过程研究[J]. 计算机工程与应用，2003，28：103～105

17.2 Web 工程

从现实中来看，Web 应用的开发与管理远不如软件工程规范。在设计开发 Web 应用系统中，往往对系统开发和设计不够重视，开发具有随意性，不能按期完成制作，系统建成后不能使客户满意，修改不断，耗费很多人力和资金，费用超出预算等。这些问题的出现很大程度上是因为随着 Internet 和 Web 技术的快速发展，大量的新型商务应用开始基于 Web 开发，同时很多的传统应用开始向 Web 环境移植，但是开发人员的方式还停留在 Web 发展的早期阶段。开发人员通常用一种粗糙的、随意的方式开发，缺乏严格的过程、有效的方法、严密的技术和相应的质量保证机制。在开发、发布、实施和维护 Web 的过程中，可能就会碰到一些严重的问题，失败的可能性很大。一个项目的失败将可能导致很多问题。当这种情况发生时，人们对 Web 和 Internet 的信心可能会无法挽救地动摇，从而引起 Web 危机，并且，Web 危机可能会比软件开发人员所面对的软件危机更加严重，更加广泛。正如软件危机的出现促使人们研究软件工程一样，为保证项目的开发进度和质量，人们不得不将注意力投入到研究如何规范地开发 Web 应用的方法上。那能否将软件工程的原理、概念和方法应用到 Web 开发中呢？由于 Web 应用开发与传统软件开发不同，传统软件工程方法和技术应用到基于 Web 的系统开发中，显得力不从心。在开发复杂的基于 Web 的系统中，迫切需要符合适应于开发要求的另一套独立体系下的规范。

为了避免 Web 危机的发生，在 Web 应用的开发中取得更大的成功，我们迫切需要一个严格的步骤和新方法、新工具来开发、发布和评估基于 Web 的系统。于是，就有了 Web 工程的概念。

Yogesh Deshpande 和 Steve Hansen 等人早在 1998 年就提出了 Web 工程的概念："使用合理的、科学的工程和管理原则，用严密的和系统的方法来开发、发布和维护基于 Web 的系统。[①]"概括地说，Web 工程是一门关于建立科学的、工程和管理的原则，采用系统和严密的方法来开发、实施和维护基于 Web 的应用系统的学科。

Web 工程需要进化和成熟，我们必须研究和评估现有的方法和经验，研究适合 Web 工程的软件技术和过程。这对 Web 应用的开发具有重要的现实意义和应用价值。

17.3 Web 工程技术

如今的 Web 应用程序越来越流行，基于浏览器/服务器(Browser/Server，B/S)结构的软件也日渐增多，Web 应用技术也可谓百花齐放，各有千秋。这些技术的出现，极大地丰富了 Web 应用的能力，提高了软件工程开发效率，也在一定程度上促进了 Web 应用的发展。

---

① Yogesh Deshpande and Steve Hansen. Web Engineering: creating a discipline among disciplines[J]. IEEE Software，2001，(2)：82～87

当前主要的 Web 技术可以分为 Web 服务器技术、Web 开发技术及 Web 辅助技术 3 类。

17.3.1 Web 服务器技术

Web 服务器是用来组建 Web 站点并支持 Web 应用的软件系统，其主要用途就是为企业组建站点。目前 Web 服务器软件系统有多种选择，下面对现在流行的 Web 服务器系统进行简要介绍。

1. Microsoft IIS

Microsoft 的 Web 服务器产品为网络信息服务器(Internet Information Server，IIS)，IIS 是允许在公共 Intranet 或 Internet 上发布信息的 Web 服务器。

IIS 是一种 Web 服务组件，其中包括 Web 服务器、文件传输协议(File Transfer Protocol，FTP)服务器、网络信息传输协议(Network News Transport Protocol，NNTP)服务器和简单邮件传输协议(Simple Mail Transfer Protocol，SMTP)服务器，分别用于网页浏览、文件传输、新闻服务和邮件发送等方面，它使在网络(包括互联网和局域网)上发布信息成了一件很容易的事。它提供 Internet 服务器扩展(Internet Server API，ISAPI)作为扩展 Web 服务器功能的编程接口；同时，它还提供一个 Internet 数据库连接器，可以实现对数据库的查询和更新。

IIS 变得普及的一个关键就是引入了动态服务器页面(Active Server Pages，ASP)，这是 Microsoft 用于建立动态网页的技术。ASP 支持多种脚本语言，使其可以很容易地访问其他服务器的软件组件。

IIS 具有很高的执行效率、出色的安全保密性、易于管理以及启动迅捷等特点。它既可用于集成现有的应用方式，又可用于实施 Web 应用系统。IIS 安装简单，操作方便，能够负担今日的高容量站点，有不少大型的商务站点都是建立在 IIS 之上的。

2. WebSphere 应用服务器

WebSphere 是 IBM 的集成软件平台。它包含了编写、运行和监视全天候的工业强度的随需应变的 Web 应用程序和跨平台、跨产品解决方案所需要的整个中间件基础设施，如服务器、服务和工具。WebSphere 提供了可靠、灵活和健壮的集成软件。WebSphere 应用服务器是该基础设施的基本平台，其他所有产品都在它之上运行。它是一种功能完善、开放的 Web 应用程序服务器，是 IBM 电子商务计划的核心部分。它基于 Java 的应用环境，用于建立、部署和管理 Web 应用程序。IBM 将提供 WebSphere 产品系列，通过提供综合资源、可重复使用的组件、功能强大并易于使用的工具以及支持 HTTP 和互联网内部对象请求代理协议(Internet Inter-Object Request Broker Protocol，IIORBP)通信的可伸缩运行时环境，来帮助开发人员从简单的 Web 应用程序转移到电子商务世界。

3. WebLogic

BEA 公司的 WebLogic 是用于开发、集成、部署和管理大型分布式 Web 应用、网络应用和数据库应用的 Java 应用服务器。将 Java 的动态功能和 Java Enterprise 标准的安全性引

入大型网络应用的开发、集成、部署和管理之中。

4. Apache

Apache 仍然是世界上用得最多的 Web 服务器，市场占有率达 60%左右。世界上很多著名的网站都是 Apache 的产物，它的成功之处主要在于它的源代码开放、有一支开放的开发队伍、支持跨平台的应用(可以运行在几乎所有的 UNIX、Windows、Linux 系统平台上)以及它的可移植性等方面。

5. Tomcat

Tomcat 是 Apache 软件基金会(Apache Software Foundation)的 Jakarta 项目中的一个核心项目，由 Apache、Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 公司的参与和支持，最新的 Servlet 和 Java 服务器页面(Java Server Pages，JSP)规范总是能在 Tomcat 中得到体现。Tomcat 是一个小型的轻量级应用服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试 JSP 程序的首选。因为 Tomcat 技术先进，性能稳定，而且免费，因而深受 Java 爱好者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的 Web 应用服务器。

17.3.2 Web 开发技术

1. HTML 技术

HTML 是浏览器识别的语言，通过它可以让浏览器显示出任何需要提供的信息(文本、表格、表单、图像等)，它是 Web 应用的最终结果。HTML 文件是一种静态的页面，其优点是不用经过其他的处理，而且可以被浏览器或代理服务器存在缓存中，所以对 HTML 页面请求的反应时间比较快；另一个优点就是它可以通过一些网页编辑器(如 FrontPage，Dreamweaver 等)以所见即所得的方式生成和编辑，这样就可以很方便地维护和修改。然而，它静态的特性往往不能满足需要，我们不可能为一点点的改变去创建许多 HTML 文件。另外不同的浏览器所支持的 HTML 规范是不同的，有时一个页面在一种浏览器中的布局很合适，但是到了另一个浏览器中就会产生这样或那样的问题；有时同一种功能，不同的浏览器的实现方法也会不同。多数情况下，人们都是利用各种浏览器都支持的 HTML 功能，这种解决方案势必限制了 Web 应用的表现力，影响了实现某种功能的难易程度。

2. PHP 技术

超文本预处理语言(Hypertext Preprocessor，PHP)是一种 HTML 内嵌式的语言，是一种在服务器端执行的嵌入 HTML 文档的脚本语言，语言的风格类似于 C 语言，被广泛运用。它大量地借用 C 和 Perl 语言的语法，并结合 PHP 自己的特性，使 Web 开发者能够快速地写出动态页面。

PHP 的语法和实用报表提取语言(Practical Extraction and Report Language，PERL)很相似，但是 PHP 所包含的函数却远远多于 PERL，PHP 没有命名空间，编程时必须努力

避免模块的名称冲突。一个开源的语言虽然需要简单的语法和丰富的函数,但 PHP 内部结构的天生缺陷导致了 PHP 只适合编写中小型业余网站。PHP 语法简单,易学易用,利于快速开发各种功能不同的定制网站,PHP 因为结构上的缺陷,使它在复杂的大型项目上的开发和维护都比较困难。

3. ASP 技术

动态服务器页面(Active Server Page,ASP)是 Microsoft 的 Windows IIS 系统自带的脚本语言,利用它可以执行动态的 Web 服务应用程序。ASP 的语法非常类似 VB(Visual Basic),学过 VB 的人可以很快上手,ASP 也是这几种脚本语言中最简单易学的开发语言。但 ASP 也是这几种语言中唯一的一个不能很好支持跨平台的语言。

因为 ASP 脚本语言非常简单,因此其代码也简单易懂,结合 HTML 代码,可快速地完成网站的应用程序。和 PHP 一样,ASP 简单且易于维护,很适合小型网站应用,通过分布式组件对象模型(Distributed Component Object Model,DCOM)和事务服务器(Microsoft Transaction Server,MTS)技术,ASP 甚至还可以完成小规模的企业应用,但 ASP 的最大缺点就是不支持跨平台的系统,在大型项目开发和维护上较困难。

4. JSP 技术

JSP 是 Sun 公司推出的一种动态网页技术。JSP 技术是以 Java 语言作为脚本语言的。JSP 本身虽然也是脚本语言,但是却和 PHP、ASP 有着本质的区别。PHP 和 ASP 都是由语言引擎解释执行程序代码,而 JSP 代码却被编译成 Servlet 并由 Java 虚拟机执行,这种编译操作仅在对 JSP 页面的第一次请求时发生。因此普遍认为 JSP 的执行效率比 PHP 和 ASP 都高。

JSP 是一种服务器端的脚本语言,开发效率较高,JSP 可以使用 JavaBeans 或者 EJB(Enterprise JavaBeans)来执行应用程序所要求的更为复杂的处理,但是这种网站架构因为其业务规则代码与页面代码混为一团,不利于维护,因此并不适应大型应用的要求,取而代之的是基于模型-视图-控制(Model-View-Controller,MVC)的 Web 架构。JSP 对于网站开发来讲不像 PHP 和 ASP 那样易学易用,支持 Java 的主机也少于支持 PHP 的主机,这从一定程度上限制了 Java 技术在网站上的发展,不过在企业软件应用上来讲,MVC 还是拥有相当大的优势的,虽然其配置和部署相对其他脚本语言来说要复杂一些,但对于跨平台的大中型企业应用系统来讲,基于 Java 技术的 MVC 架构几乎成为唯一的选择。

17.3.3 Web 辅助技术

1. UML 扩展机制

为了摆脱软件开发时间长、成本高、生命周期短等弊病,计算机工作者一直在不断地摸索恰当的软件建模方法。从原型式开发到现在流行的面向对象建模技术,都是为复杂应用系统建立模型。建立恰当的模型可以对系统进行抽象和简化,帮助开发者更好地理解系统。Web 应用程序规模越来越大,复杂程度越来越高,对其进行抽象和建模在整个 Web 应用开

发中显得尤为重要,而以往的开发方法和辅助工具显得苍白无力。统一建模型语言(Unified Modeling Language,UML)是一种功能强大的基于对象技术的可视化建模语言,其适用范围很广,适用于各种软件开发方法、软件生命周期的各个阶段、各种应用领域以及各种开发工具,用 UML 进行 Web 应用建模,可以达到预期的效果。

但由于 Web 应用系统中存在各式各样的表现元素,应用程序中的一些元素无法用标准的 UML 模型元素表示。为解决这一问题就必须对 UML 进行扩展,以使一些特殊的 Web 元素可以和系统中的其他模型结合起来,形成合理的抽象,建立一套与 Web 建模相适应的模型元素及表示方法。而 UML 的扩展机制为这种扩展提供了基础,满足了用户的这些需求。它可为建模者提供新的模型元素以及可附加在模型元素上的各种形式的信息。UML 的扩展机制是 UML 的基本组成部分,它说明怎样用新的语义来定制、扩展 UML 的模型元素。通过扩展机制,用户可以定义和使用自己的元素,或将已定义的元素专有化,以清晰、精确地定义一些概念或表达一些模型。

UML 有 3 种核心扩展机制,包括构造型、标记值和约束。其中最重要的扩展机制是构造型,可适用于所有类型的建模元素。它是一种在已定义的模型元素的基础上构造一种新的模型元素的机制。这样构造出来的新的建模元素就称为构造型的建模元素,被扩展的已定义元素称为它的基元素。一个构造型元素不能改变基元素的结构,但可添加某种新的语义。作为 UML 的一种基本表示法,构造型可用一个带有一对双尖括号的词组来表示,如《navigation》。约束是 UML 中限制一种或多个元素语义的规则,定义了模型如何组织在一起,通常用一对"{}"之间的字符串表示。标记值是附属于 UML 元素的性质,可以与一个元素相关联,允许在建模的任何元素上加标注,用来增加模型元素的语义,通常用带括号的字符串表示。

2. Web 服务

Web 服务是为了让地理上分布在不同区域的计算机和设备一起工作,以便为用户提供各种各样的服务。用户可以控制要获取信息的内容、时间及方式,而不必在无数个信息孤岛中浏览,去寻找自己所需要的信息。显然 Web 服务将成为下一代 Web 的主流技术。利用 Web 服务,公司和个人能够迅速且廉价地通过互联网向全球用户提供服务,建立全球范围的联系,在广泛的范围内寻找可能的合作伙伴。随着 Web 服务技术的发展和运用,我们目前所进行的开发和使用应用程序的信息处理活动将过渡到开发和使用 Web 服务上。将来 Web 服务会取代应用程序成为 Web 上的基本开发和应用实体。

Web 服务是在现有的 Web 技术和设施之上,通过制定新的协议和标准、提出新的技术来实现的。新提出的与 Web 服务相关的主要协议和技术包括简单对象访问协议(Simple Object Access Protocol,SOAP)、Web 服务描述语言(Web Services Description Language,WSDL)及统一描述、发现和集成(Universal Description,Discovery and Integration,UDDI)。SOAP 用来定义数据描述和远程访问的标准;WSDL 是发布和请求 Web 服务的描述语言;UDDI 则把 Web 服务与用户联系起来,起中介作用。当然,Web 服务的具体实现并不局限在这几种协议和技术上,任何支持 Web 标准的系统都能支持 Web 服务。

Web 服务具有以下一些主要特性。

(1) 互访性。Web 服务通过 SOAP 实现相互间的访问，任何 Web 服务都可以与其他 Web 服务进行交互，避免了不同协议之间的相互转换。Web 服务可以用任何语言编写，因此开发者不需要更改开发环境就能开发新的 Web 服务，同时还可以在新的 Web 服务中使用已有的 Web 服务，而不必考虑 Web 服务的实现语言、运行环境等具体实现细节。例如，用 Delphi 编写的 Web 服务可以使用由 Visual C 编写的 Web 服务，反过来也可以。

(2) 普遍性。Web 服务使用 HTTP 和 XML 进行通信，任何支持这些技术的设备都可以拥有和访问 Web 服务。Web 服务不仅在计算机网络上出现，而且将在电话、汽车、家用电器等设备中出现。现在，各主要设备和软件供应商都已宣布支持 SOAP 和周边 Web 服务技术，相信在未来，Web 服务将普遍存在于社会生活中的各个领域。使用 Web 服务，人们就能够通过网络在异地指挥家中的电器设备工作，进行诸如煮饭、加温、降温等操作。

(3) 廉价性。Web 服务供应商提供的免费工具箱能够让开发者快速创建和部署自己的 Web 服务，这就降低了 Web 服务的开发费用，同时也加快了开发速度。

17.4 Web 工程的层次

Web 工程是用系统的、严密的、可以测量的方法来开发、实施和维护基于 Web 的应用。显然 Web 工程包含了程序设计和软件开发，这样就不可避免地会采用某些软件工程的思想和方法，因此 Web 工程和软件工程是紧密联系的。基于 Web 的系统和应用的开发包括专门的模型、适合 Web 应用开发特点的软件工程方法，以及一组使其奏效的重要技术。过程、方法和工具提供了 Web 的分层方法，在概念上与传统的软件工程层次相同，如图 17-6 所示。

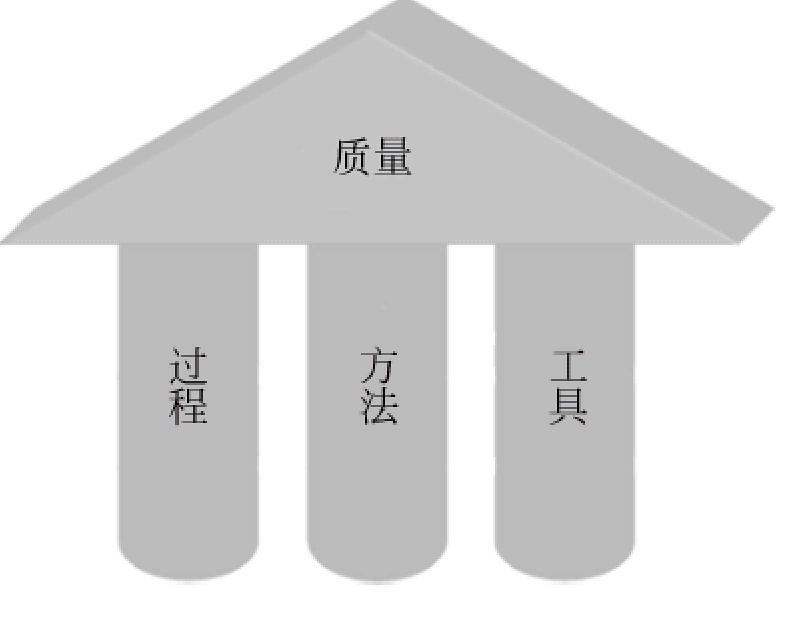


图 17-6 Web 工程层次图

Web 工程又是不同于软件工程的，不是软件工程的完全复制，而是借用了很多软件工程的基本概念和原理。

17.4.1 质量

获得高质量的 Web 应用系统是 Web 工程的目标，如何认识 Web 应用系统的质量呢？Olsina 和他的同事设计了一个“质量需求树”[①]，定义了一组可产生高质量 Web 应用的技术属性，包括功能性、可用性、可靠性、效率和可维护性，如图 17-7 所示。这些特性为评估 Web 应用系统的质量提供了一定的基础。

---

① Olsina，L et al.. Specifying Quality Characteristics and Attributes for Web Sites. Proc 1st ICSE Workshop on Web Engineering，ACM，Los Angeles，1999，5

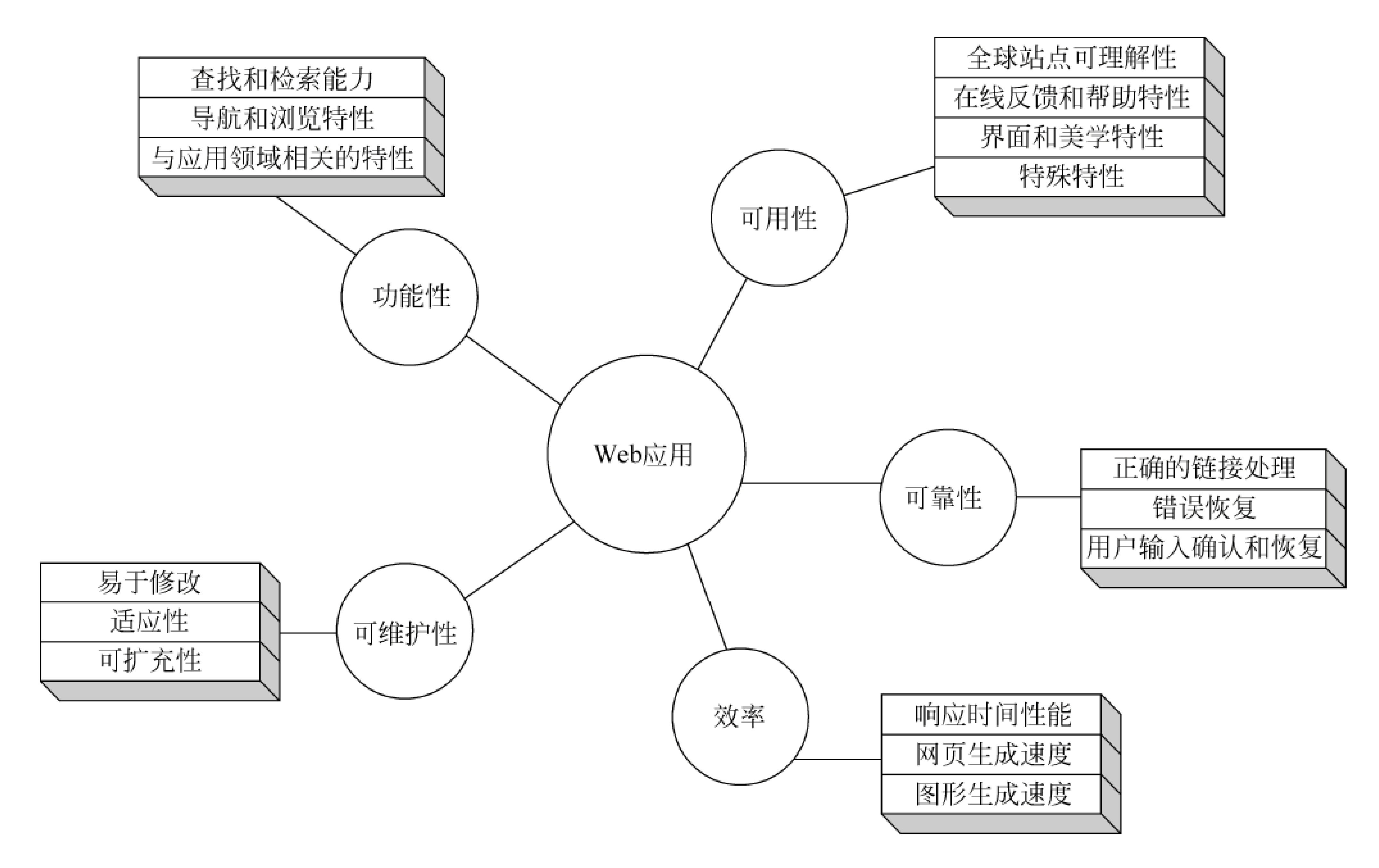


图 17-7 Web 质量需求树

17.4.2 过程

传统软件开发过程的主要组成部分是开发周期模型，一个定义良好的软件开发周期模型，可以使软件开发可控。由于 Web 应用自身的特点，传统典型的几种生命周期模型不适于指导 Web 开发，指导 Web 应用开发的过程模型需要考虑到以下几个方面。

(1) 迭代开发思路。Web 应用从本质上就是迭代和渐增的。在这里反馈环路是必需的标准结构。开发过程就是一次次的迭代反复过程。随着迭代的进行，系统的功能不断完善。

(2) 快速原型方法。Web 应用最基本的特性就是其易用性和演化性，是高度人性化的系统。这是和传统软件最大的区别。反映在开发中就是要重视界面和可操作性，并提高到一个重要的高度。提前开发原型就成了当然的选择。

(3) Web 应用的开发通常具有明确的技术分工，如专门的外观设计人员负责信息的布局和显示，专门的技术人员负责网页的组装和导航，信息专家负责应用领域的信息结构和组织，这也要求描述模型应该显式地表述为不同的层次。

(4) 超文本特性(链接和导航)是 Web 应用独有的特征，在较大规模的 Web 应用开发中，对 Web 导航的建模已经成为复杂的问题。网页已经从以前简单的文本、图形、超链接发展到动态的 Web 网页以及嵌入各种程序的客户端网页，但软件工程并未对导航建模提供相应的支持。混乱的导航结构往往引发 Web 应用的耦合性增加、体系结构过于复杂以及扩展和维护工作等诸多问题，因此建立合理的导航模型是成功开发 Web 应用的必要条件。

(5) 符合 Web 应用特点的项目管理方法。软件开发离不开项目管理，只有将开发过程和项目管理过程结合起来，才能增进工程过程的可控性和规范性，约束工程范围和进展，提高工程结果的可用性。Web 应用的项目管理更加重视人的组织和管理，重视分析和设计等

前期工作，重视迭代过程的交付和平衡。

考虑到上述问题，我们提出 Web 开发一般过程，如图 17-8 所示。这个过程模型应该能帮助开发人员注意 Web 应用系统的复杂性，降低开发风险，处理变更的可能性，快速发布 Web，当项目进行时能为管理提供反馈，而且整个开发过程必须是可监督的和可跟踪的。

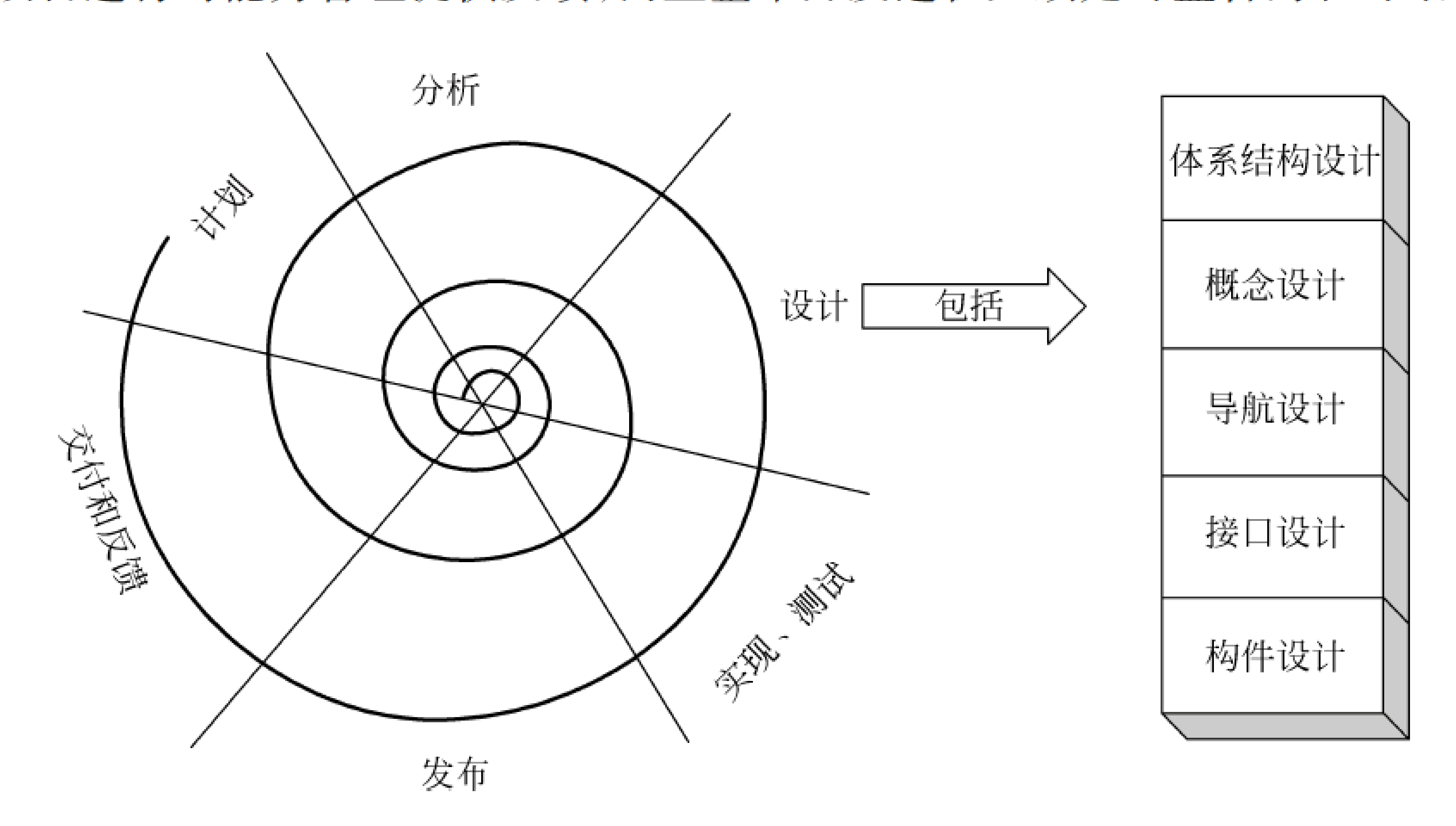


图 17-8 Web 开发一般过程

17.4.3 方法

随着 Web 越来越多地被集成到小公司和大公司的业务策略中，对构造可靠的、可用的和自适应的系统的需求变得越来越重要。因此，有必要在 Web 应用中使用规范化的方法。与传统的应用系统开发相比，Web 应用开发缺乏系统化的、结构化的方法和模型支持。

目前，Web 应用设计一方面随着 Web 应用程序由以文档为中心向以应用为中心发展，以往简单的基于页面的开发方法已经不再适用，人们迫切需要采用系统化的方法开发 Web 应用程序；另一方面由于 Web 应用程序和传统应用程序相比有着本质区别，传统的软件工程方法在 Web 应用程序的开发中局限性太大。为解决该问题，Web 应用的工程化方法已经成为当前 Web 应用开发研究的热点，其目标是提供全面支持 Web 应用开发生命周期的模型和方法。Web 开发方法的目标是系统化 Web 应用的开发全过程，提高开发效率和开发质量。为达到此目标，一个成熟的 Web 开发方法应该具备以下的主要特征或功能。

(1) 易于掌握。开发方法的一个目的是为了减轻开发人员的工作量，所以应该具有易于掌握的特点，但这不应以牺牲表达能力为代价，即 Web 开发方法应该能对 Web 程序的各个方面进行全面详细的描述。

(2) 对复杂系统建模的能力。Web 应用的范围涵盖从简单的静态站点到动态交互的 Web 应用。近年来 Web 应用发展迅速，大量传统信息和数据库系统被移植到 Web 环境下，一种新型的 Web 应用程序出现了，这些程序利用 Web 平台支持、执行商业过程以及工作流，例如，出租和预订服务、虚拟拍卖或在线保险等。成熟的 Web 开发方法应该能适用这种需求，这就需要有对商业过程、工作流进行建模的能力，并和 Web 系统设计的其他部分有机结合。

(3) 表现层建模的能力。传统的设计方法一般不很重视界面设计，而和传统的软件系

统相比，Web 系统表现设计有自己的特点：①系统的很多高级功能体现在表现层，这需要提高界面的设计质量；②界面中包含着大量的多媒体信息；③Web 设计方法一般不仅给设计人员使用，而且需要给美工、编辑等使用，他们更关心系统的表现设计。所以 Web 开发方法需要能针对这些特点对表现层建模。

(4) 系统定制的支持。Web 系统成功与否主要依赖于用户的满意程度。成功的 Web 系统应该具有丰富的功能，易于使用的界面和定义良好的导航结构。而为了达到更高的用户满意度，一个主要的技术是通过个性化定制把合适的内容在合适的时间分发给合适的人。开发方法需要提供系统定制能力，这主要通过对用户的定义和描述来完成，其中包括对用户分组以及用户之间联系的处理。

(5) 模型集成和连通的能力。能够在较高的抽象层次上表达系统和资源是怎样集成的。一方面，在很多组织中新开发的 Web 应用系统需要和以前存在的业务系统密切关联。这些业务系统可能在不同的平台和实现语言下开发。开发方法应该能支持和这些遗产系统无缝连接；另一方面，组件的集成大部分依赖于接口描述，开发方法应提供精确的和无二义的对组件接口建模和文档化的能力；最后，Web 应用系统需要和大量的资源及信息服务等相连接，这些可能不局限于组织内部，开发方法应提供表达和存取机制。

(6) 自动生成能力。能否提供自动生成能力以及能力的大小是 Web 开发方法成熟度的一个重要衡量标准，理想的开发方法应能提供从模型描述到运行、配置代码的自动生成。

(7) 工具和文档支持。理想的工具应能支持在用户参与下，完成从需求分析到实现维护的整个开发过程。丰富的文档支持是设计者能否掌握开发方法的重要方面。可以说工具和文档的支持能力是开发方法能否得到广泛应用的关键。

近年来，一些系统化开发 Web 应用的方法应运而生，这些方法来源于不同领域的研究，如来源于数据库领域的研究方法 RMM(the Relationship Management Methodology)，来源于多媒体研究领域的方法 HDM(Hypermedia Design Method)、WebML(Web Modeling Language)，以及来源于面向对象研究领域的方法 OOHDM(the Object-Oriented Hypermedia Design Method)、UWE(the UML-based Web Engineering)等，都是借助传统软件工程中的面向对象或结构化模型，支持 Web 工程的分析与设计。尽管 Web 应用开发的各种方法有所不同，但仍然存在一些相同的关注点和开发步骤。这些 Web 应用开发方法的共同点在于一般将 Web 系统模型分为概念模型、导航模型和展示模型，经过概念建模(集中在描述问题域，系统要做什么，独立于任何技术细节)、逻辑建模(集中于系统的操作，隐藏了针对特定平台的实现细节)、物理建模(适应应用的逻辑模型，获得针对选择平台实现的详细说明)和实现 4 个过程完成 Web 系统开发。概念、导航、展示模型分别描述系统的一个不同侧面，可以看成是 Web 模型的不同视图。概念模型描述 Web 应用的领域对象及其关系，是导航模型的基础；展示模型描述 Web 页面展示形式，是导航对象和导航行为的最终体现；而导航模型是 Web 模型区别于传统系统模型的重要部分，描述了 Web 应用的导航特性，并起着衔接领域模型和展示模型的作用。图 17-9 表现了这些共同的特征[①]。

① Damiano Distante, Paola Pedone, Gustavo Rossi and Gerardo Canfora . Model-Driven Development of Web Applications with UWA, MVC, and JavaServer: Faces

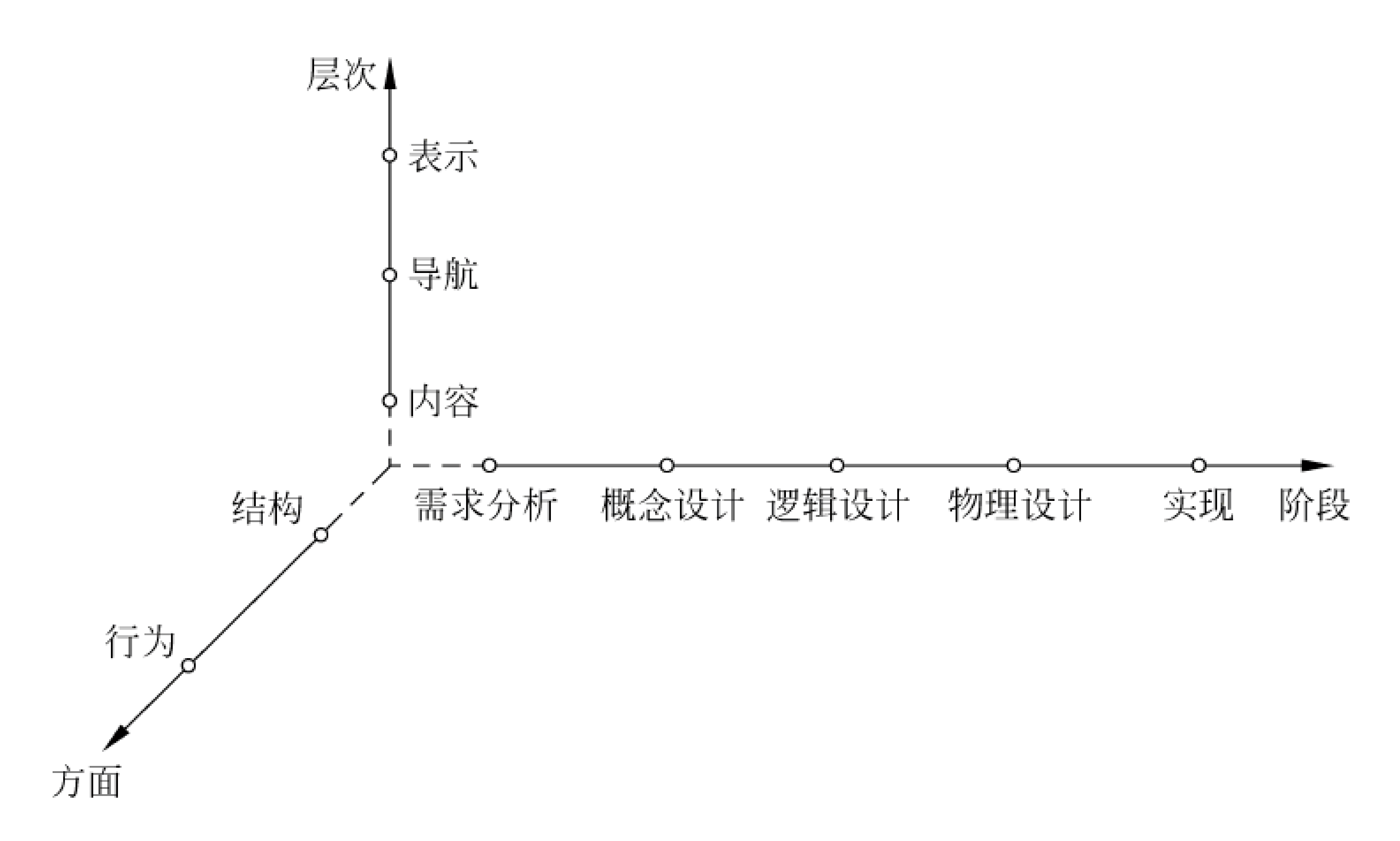


图 17-9 Web 应用程序的一般开发框架

17.4.4 工具

随着 Web 应用逐渐成熟及普遍，已经涌现出大量的 Web 开发工具和技术。Web 开发工具随着它们所支持的 Web 应用程序的大小、类型和生命周期的覆盖情况变化[①]，就应用的大小而言，开发工具应可支持个人网页、Web 站点或由多个站点组成的分布式 Web 应用；就 Web 应用的特性而言，开发工具应可能支持静态超文本应用（主要特征是静态链接和静态页面）、以数据库为中心的信息系统或动态交互的应用，并且开发工具应能支持 Web 应用生命周期的不同阶段，如图 17-10 所示。

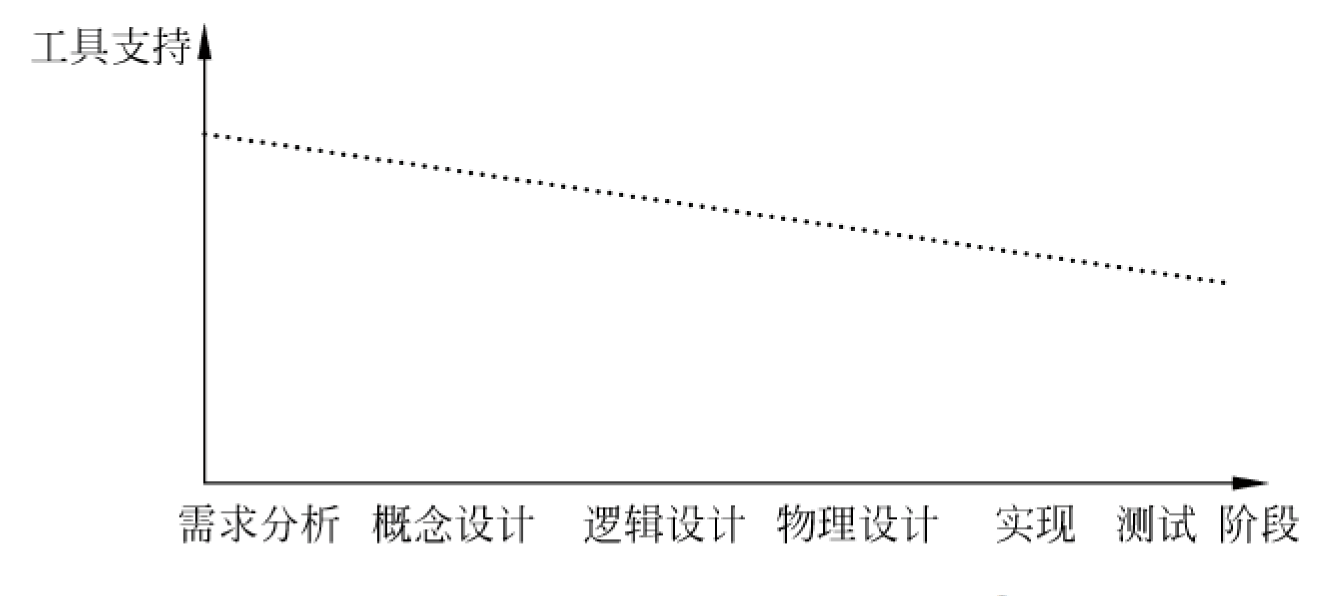


图 17-10 Web 应用开发工具需求[②]

17.5 Web 工程过程

为了说明 Web 应用开发过程，下面使用一个在线会议论文评审系统作为例子。该系统支持为某会议选择论文的全过程（包括作者提交，程序委员会的评审和选择等）。允许一般

① Hans-Werner Gellersen. Robert Wicke, Martin Gaedke Web Composition: An Object-Oriented Support System for the Web Engineering Lifecycle

② Nanard J., Nanard M.. Hypertext design environments and the hypertext design process. Communication of the ACM,1995,38 (8): 49～56

用户浏览会议信息；用户在注册成为作者登录系统后可以提交论文，可以修改提交内容，可以查看他提交的论文清单；程序委员会主席（以下简称主席）监督整个会议论文的提交和评审过程，包括创建会议（假设该系统只处理一个会议，主席已经注册，有相应的用户名和密码）、更新会议信息、预先注册程序委员会成员（以下简称成员）和评审者、给成员指定论文、改变提交论文的征文方向、给出论文的结果（接受，拒绝或未确定的）；主席能浏览所有会议相关的信息，包括论文清单、接受论文清单、拒绝论文清单、作者清单、成员清单、评审者清单、评审意见等；成员能预先注册评审者，给评审者指定论文，能浏览指定给他的论文清单和评审信息；评审者可注册，能下载指定给他的论文，提交评审意见，修改评审意见。

17.5.1 分析

Web 应用系统的需求分析是很重要的活动，需要一个系统而严密的方法。根据 Web 特性和 Web 应用的特定需求，需要采用更为开放、灵活的需求分析方法，与传统软件过程的分析不同，Web 分析阶段不但要分析 Web 本身的功能和性能，还要对可能的用户群体进行分析和调查，然后从用户的角度来开发交互的用例。

1. 用户层次

在评审系统的文本描述需求的基础上，我们识别出用户可执行以下角色：主席，成员，评审者，作者以及一般的访问用户。这些实际上就是用例模型的参与者。其用户层次如图 17-11 所示。

在图 17-11 中，评审系统访问用户处于用户层次的顶端，代表了最一般的用户类型，可以浏览会议信息，并在下面的层次中对其进行细化。其子类包括以下几个。

（1）作者：注册用户，要在截稿日期前投稿等。

（2）主席：负责创建会议，预先注册成员，给成员指定评审的论文，标记论文状态等。

（3）评审者：评审论文。

（4）成员：预先注册评审者，给评审者指定论文。

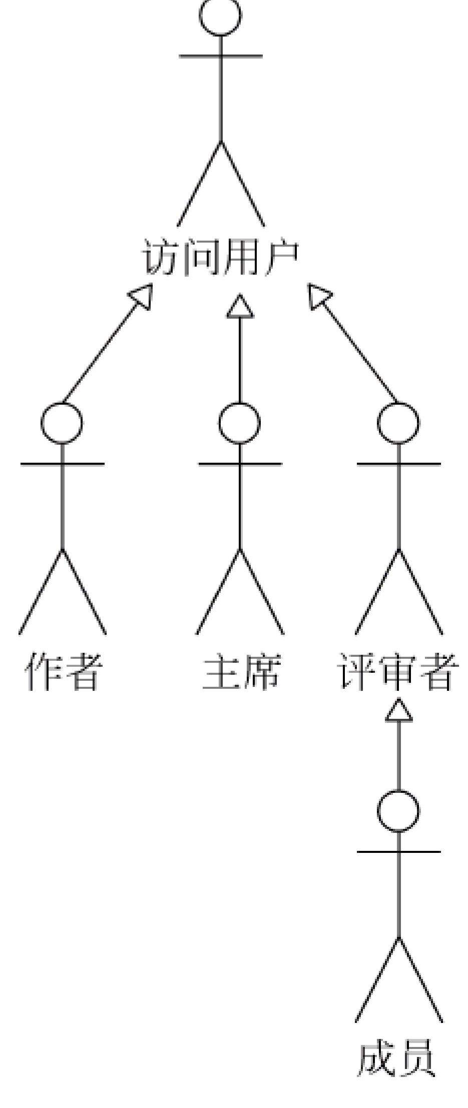


图 17-11 会议评审系统的用户层次

2. 开发用例

使用用例是功能需求的参考建模技术。一个 Web 应用的总体功能用多个用例来建模，描述了用户层次图上的每一种用户种类对 Web 应用的期望功能。用例模型将为后续的建模提供一个起点。Web 应用需求的一个独有特点是导航功能（允许用户在超文本中导航）。UWE 中使用《navigation》原型来指示功能用例和导航用例之间的区别。

评审系统的部分用例模型如图 17-12～图 17-14 所示，图中的每一个椭圆代表一个用例，描述了各个参与者和 Web 应用之间的一次具体交互。

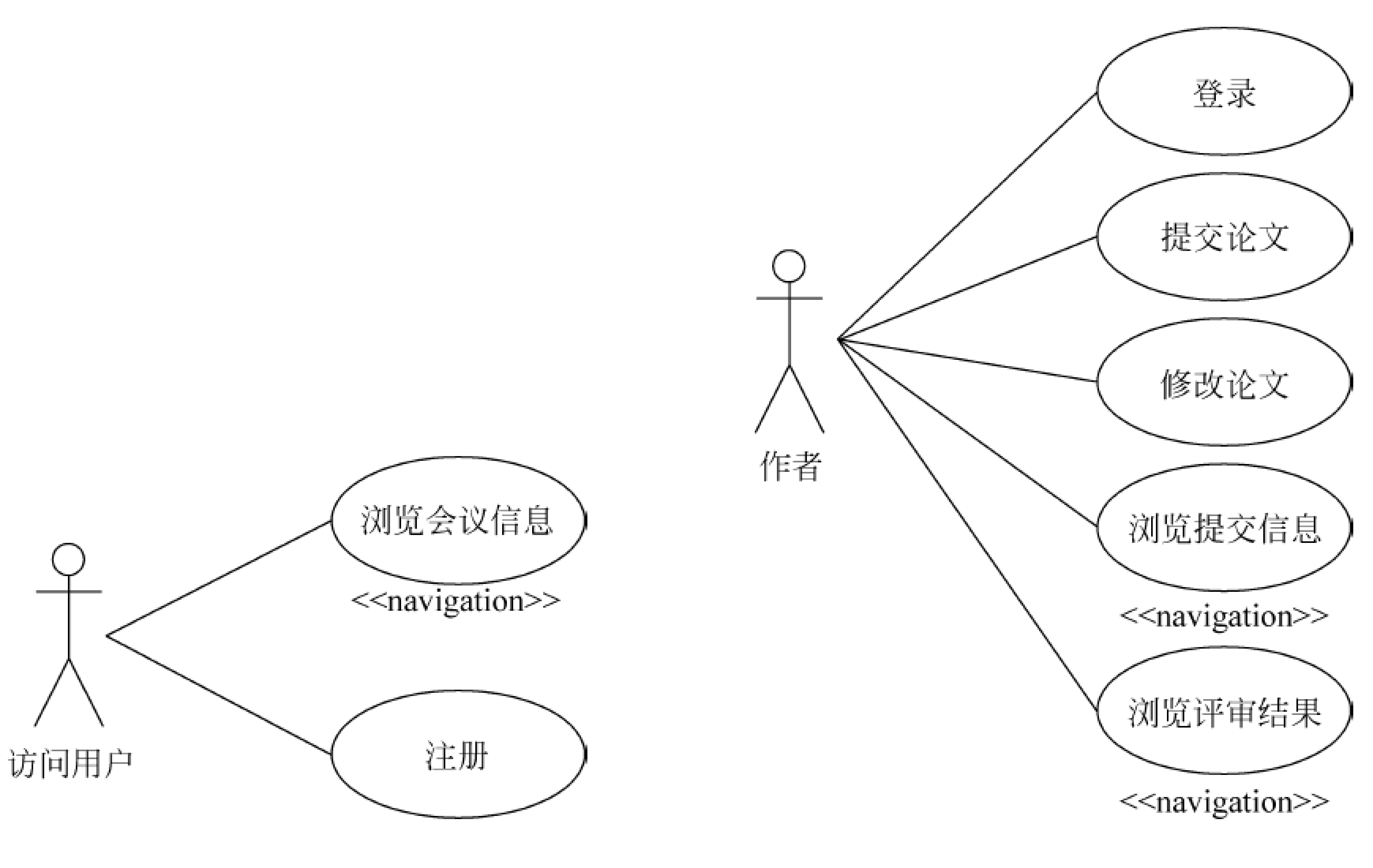


图 17-12　访问者用例图　　　　图 17-13　作者用例图

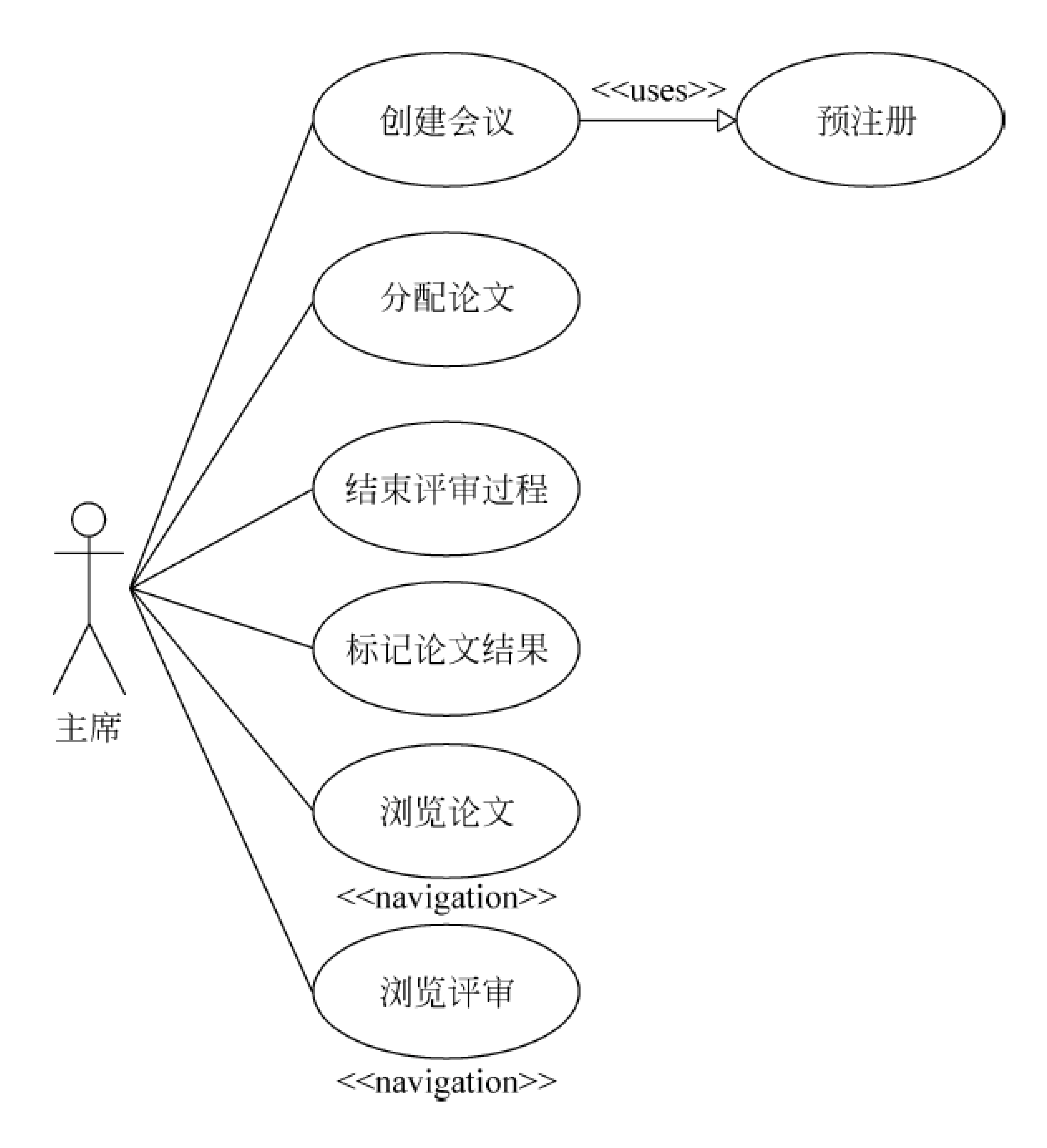


图 17-14　主席用例图

我们能用文本形式或行为图(如活动图)的方式描述每一个用例的细节。当用例是基于复杂的应用逻辑时,可以使用活动图来补充其功能需求的细节,这一部分可参考面向对象软件工程相应章节的内容。

17.5.2 设计

1. 体系结构设计

通过前面的步骤已经明确了系统的主要功能,在系统的进一步设计之前,需要确定系统将要采用的体系结构,这对以后的系统设计工作非常重要,决定将要按照什么样的方式对系统进行构造。

基于B/S架构的Web应用软件通常采用MVC框架,它要求商业模型、前端表现及有效控制分层组织,采用哪种框架组织站点并不影响对软件系统的功能需求分析,但它会影响到Web建模的描述。MVC模式的处理过程可以简单地表述为:控制器接受用户的请求,决定应该调用哪个模型进行处理,被选中的模型用事先确定的业务逻辑处理用户的请求并返回处理结果,最后控制器用相应的视图返回数据,通过表示层呈现给用户。其基本架构如图17-15所示。

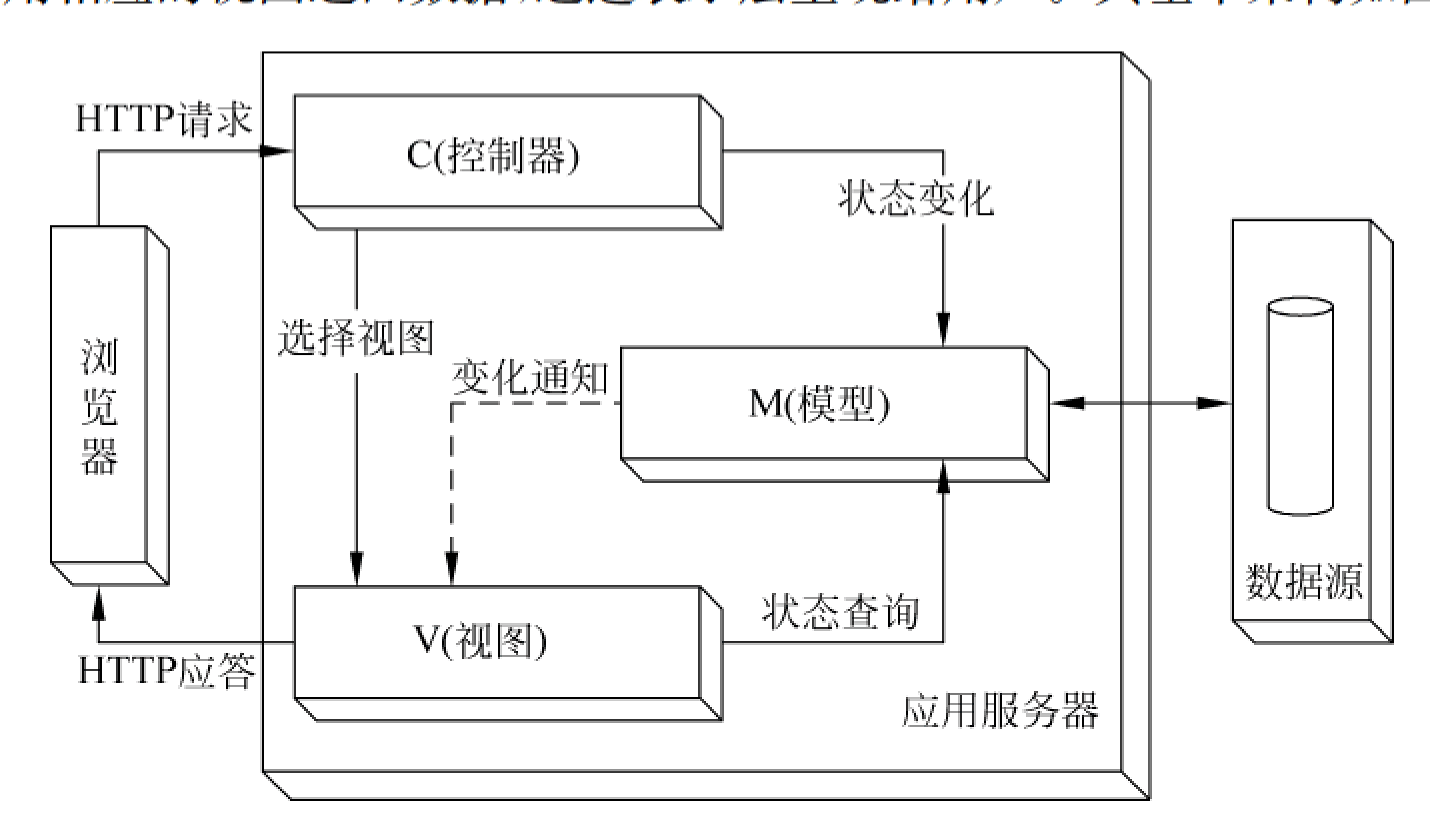


图17-15 MVC结构

(1) 视图。视图是用户看到并与之交互的界面。对Web应用系统来说,视图可以由HTML元素构成,也可以由其他一些技术如XML、XHTML等来构建。MVC可以为系统的一个应用建立很多不同的视图,在视图中没有真正的业务流程的处理,它对数据的操作大多只是数据的采集和显示,业务流程的处理由模型来完成。

(2) 模型。模型表示企业数据和业务规则,负责业务流程的处理。在MVC的3个部件中,模型拥有最多的处理任务。业务流程的处理过程对其他部件来说是暗箱操作,模型接收视图请求的数据,进行相应的处理并返回最终的处理结果。被模型返回的数据是中立的,也就是说模型与数据格式无关,采用这种机制使一个模型能为多个视图提供数据。由于应用于模型的代码只需要写一次就可以被多个视图重用,所以减少了代码编写的重复,减少了系统开发的工作量。

(3) 控制器。控制器接收用户的输入并调用模型和视图对数据进行处理,共同满足用户的请求。控制器接受请求后,并不处理业务信息,而是根据用户请求选择相应的模型,把用户的信息传递给模型,告诉模型做什么;处理完相应的数据后,模型把结果交给控制器,控制器选择符合要求的视图返回给用户。

2. 概念设计

通常用用例来捕获需求。这是一种以用户为中心的技术,能定义出应用中的参与者,并

提供一种直接的方式来实现每个参与者要求的功能。概念设计建立在用例的基础上，其目标是将由需求工程得到的信息和功能需求转变为一个域模型。常用的面向对象的建模活动如找到类、定义相关的属性和操作、定义类之间的关联、定义继承层次、找到依赖、定义接口及定义约束等将在这一阶段执行。这些活动的结果就是问题域的一个类图模型。类由属性和操作来描述，并通过 UML 标记来表示。在这个步骤中定义的类和关联将用于导航设计中。

论文评审系统的部分概念模型如图 17-16 所示。该图表明一个会议有多个主题，作者能提交多篇论文，一篇论文最多由 3 名评审家评审。

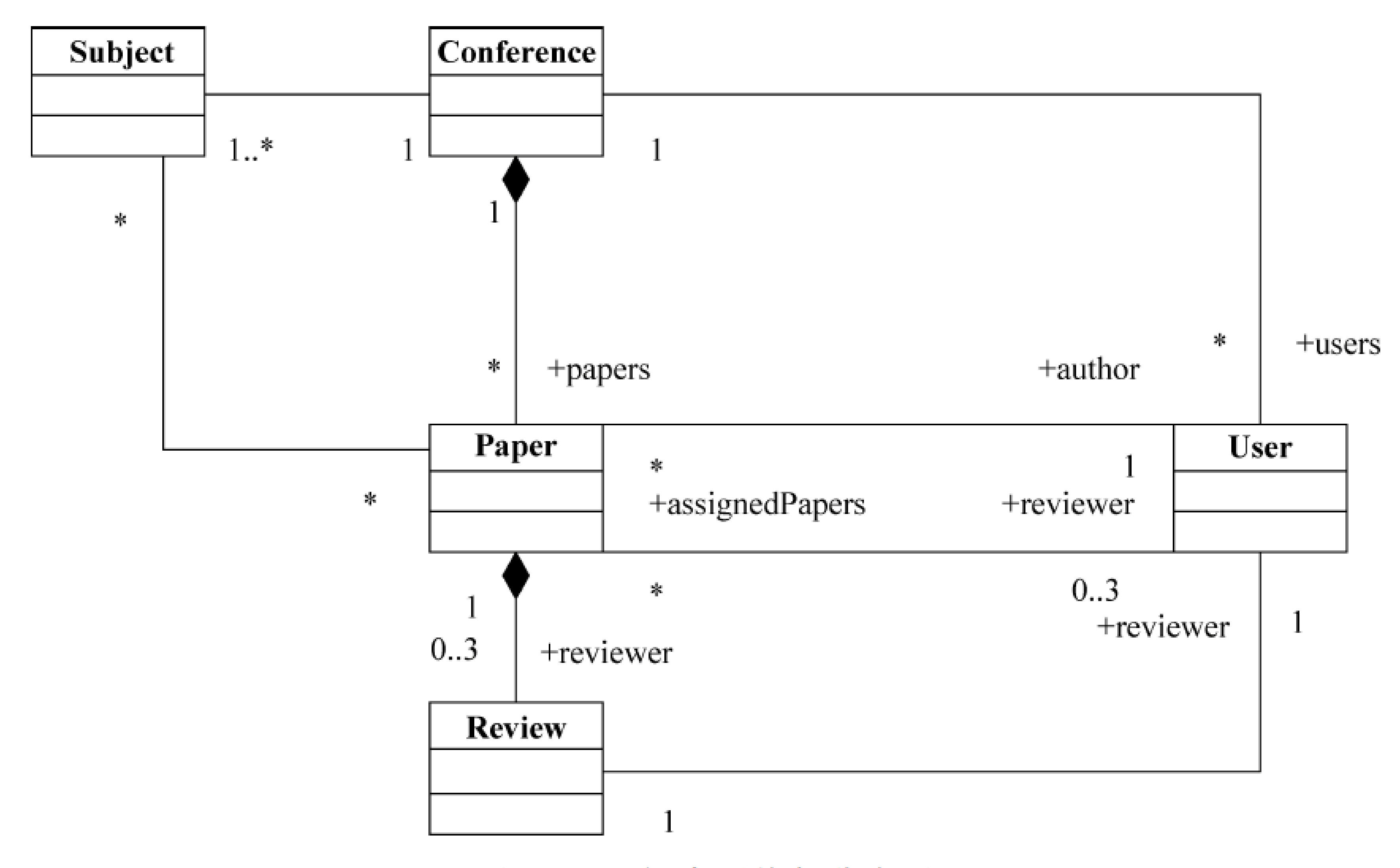


图 17-16 评审系统部分类图

作为对类图的补充，图 17-17 显示了在评审系统中论文的各种不同的状态。它显示了在提交日期前提交的论文将被指定给评审者评审，评审的结果是录用(接受)或者拒绝。

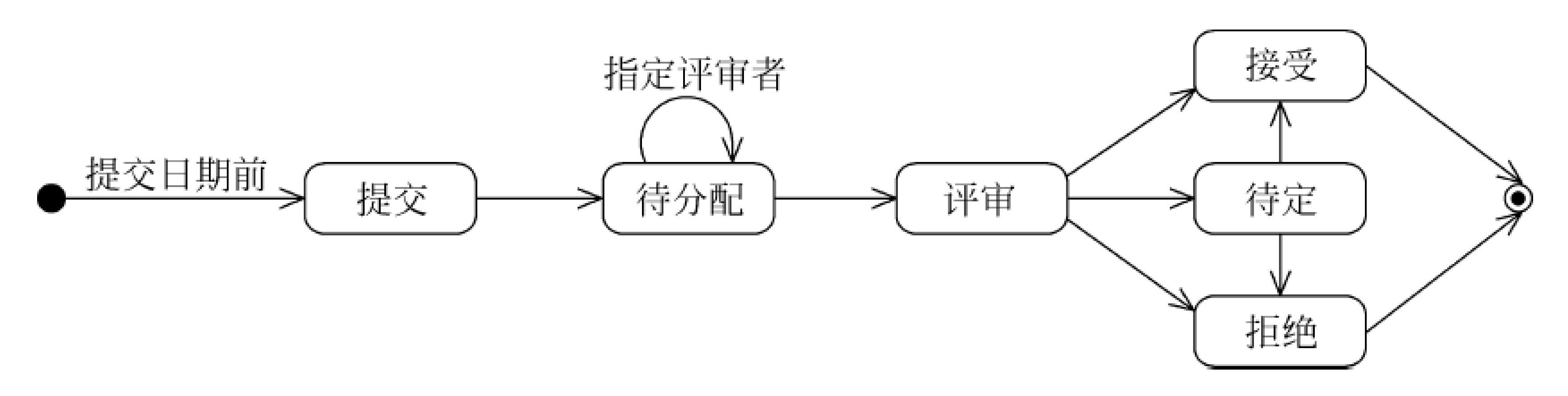


图 17-17 论文状态图

3. 导航设计

导航是 Web 应用系统成功的一个重要因素，它对系统的可用性和用户界面的友好程度有重要的影响。对 Web 环境下导航的描述应该有以下要求：首先因为 Web 导航比较复杂，因此导航信息应该从其他模型中分离出来，否则会导致模型过于复杂；其次导航模型不但要考虑静态的网页链接，还要能描述动态网页之间的链接；最后要考虑不同的激发事件类型[①]。导航设计的目标是为用户设计合适的导航路径，使他们可以访问 Web 应用系统的

① 郭小涛. 支持 Web 软件用户界面自动生成的交互模型. 2005

内容和功能。为完成这一目标，设计者首先应该为不同的用户确定相应的导航路径；其次定义如何实现导航。

（1）为不同的用户确定导航路径

在进行导航设计时，应先考虑用户层次和为每一类用户创建的相关用例，因为每一类用户使用Web应用系统的方式是不同的，因而会有不同的导航要求，导航路径自然也就不同。要利用在概念设计中得到的类和关联来建立符合系统需求的导航路径和链接，使用户能以恰当的方式与Web应用系统交互。

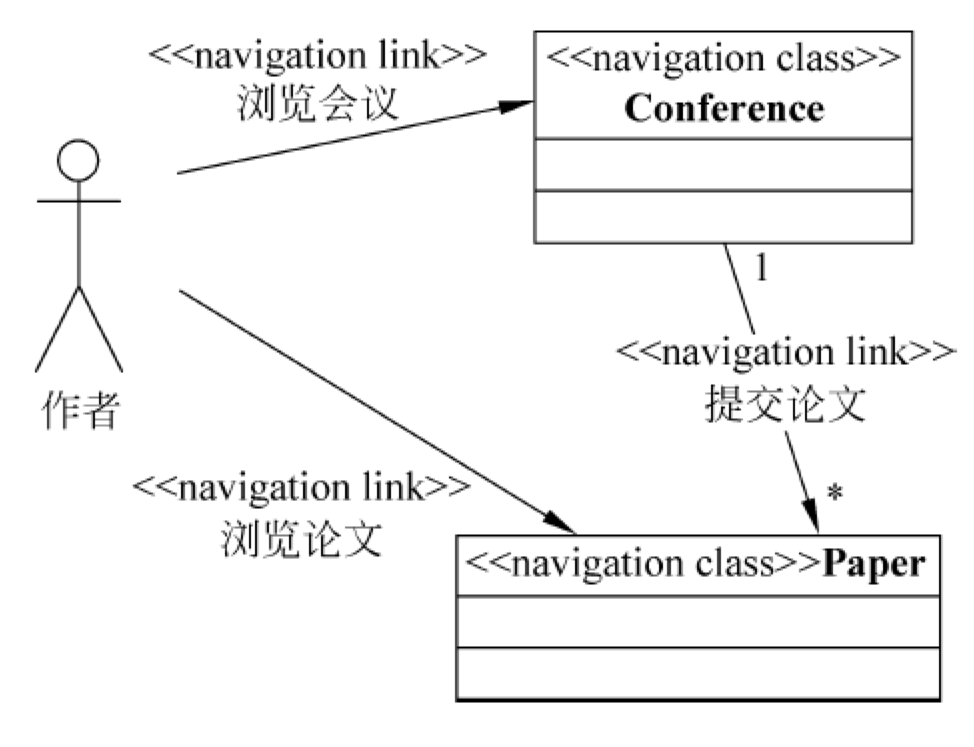


图 17-18 作者的导航路径

图 17-18 显示了作者的导航路径，从会议评审系统的主页开始，作者只能导航到他自己提交的论文处。图 17-19 显示了主席的导航路径，允许管理会议信息和论文，因此能浏览所有提交的论文、访问录用或拒收论文的清单、评论、评审者等。UWE 中使用构造型《navigation class》和《navigation link》来表示导航类和导航链接，前者表示的是概念模型中能被用户访问到的类，这些就作为导航结点；后者可由概念模型中的类之间的直接关联推导出。

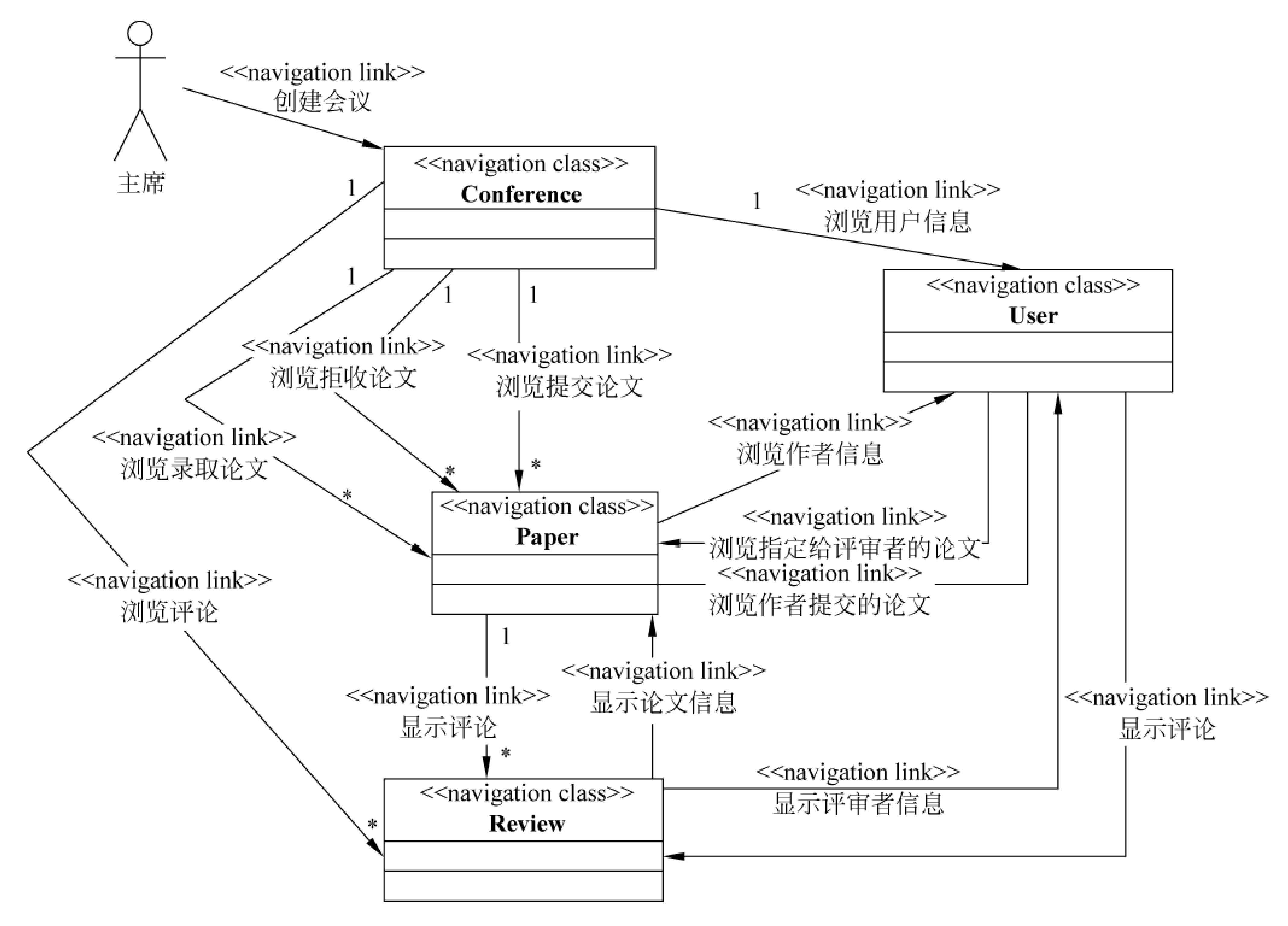


图 17-19 主席的导航路径

(2) 定义实现导航的机制

如图 17-18 和图 17-19 所示的导航路径本身还不足以描述结点间如何通过导航来访问,因此需要说明导航的实现机制。我们将用访问结构的形式来定义该机制,也就是描述导航如何通过索引、向导、查询和菜单等访问结构支持。UWE 使用构造型如《menu》、《index》、《query》、《guided tour》来说明菜单、索引、查询和向导等访问结构。

在评审系统的例子中,如果主席要查看某一篇录用论文的信息,他必须先导航到这篇指定的论文,能通过论文编号或标题搜索,也可以列出所有录用论文列表的形式实现。为导航支持的选择列表也称为索引,能允许用户从同类的一列对象中选择出一个对象。通过菜单能访问不同类型的结点;向导允许用户顺序地浏览多个结点;查询允许用户搜索结点,更多的有关访问结构的内容请参考相关资料。图 17-20 显示了加入访问结构后的主席导航路径。

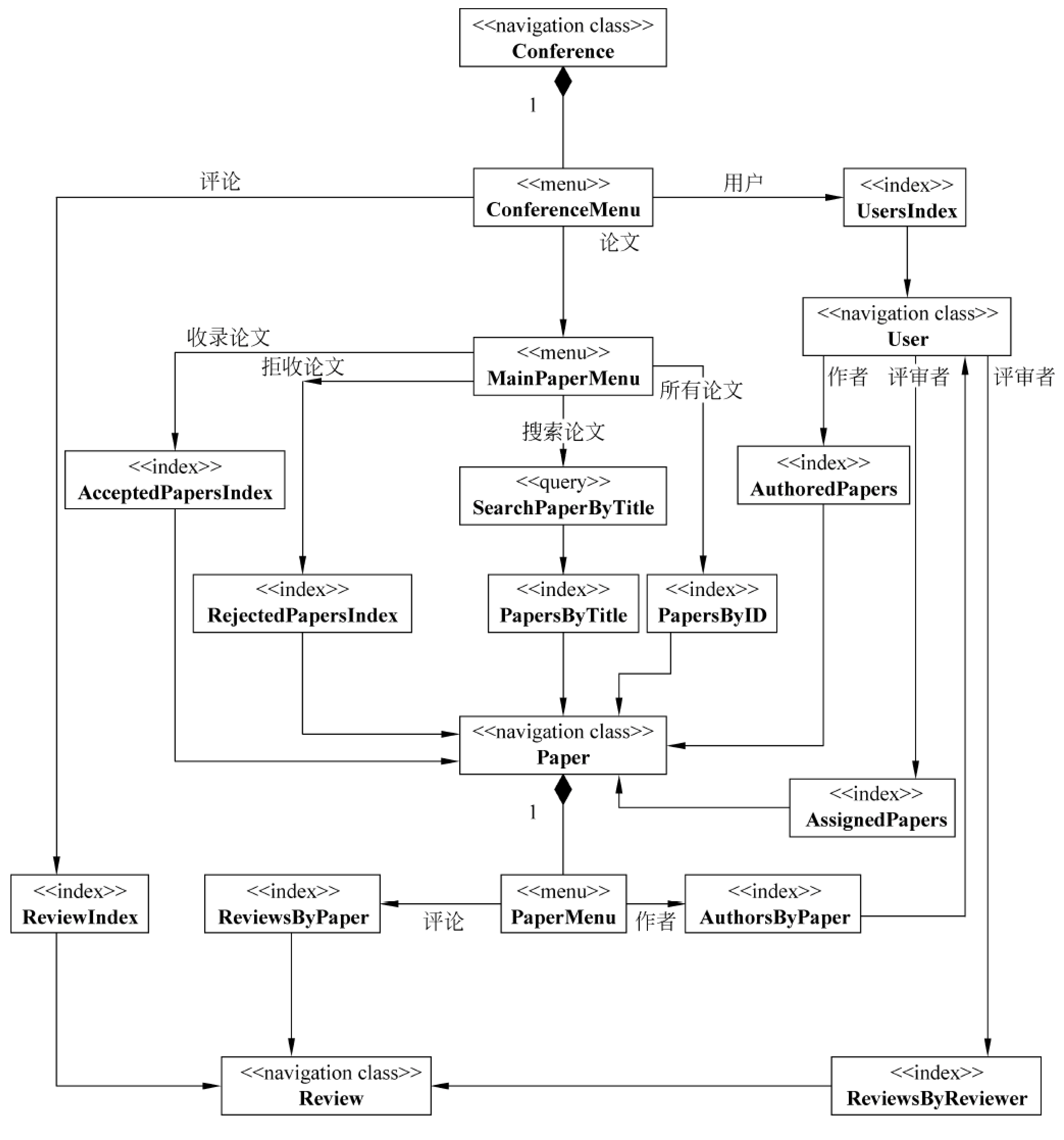


图 17-20 主席的导航路径实现

4. 表示设计

表示设计的目标在于设计用户界面的结构和行为，以保证用户和 Web 应用的交互简单明确。在这一阶段中应该要设计每个页面的组成和页面内的字段、文本框、图片、表格等；此外还要描述用户界面的行为方面，如单击一个按钮后会激活应用逻辑的哪个功能。

图 17-21 和图 17-22 显示了评审系统的 2 张表示页面的组成。PaperPage 页面中显示的是论文的一些基本信息(包括论文标号、提交日期、标题、摘要、主题等)和一个指向全文及作者的链接，此外还有一个提交评论的按钮。AuthorPage 页面有两个表示单元，即作者列表和每个作者的详细信息。在这 2 张图中使用构造型《page》和《presentation unit》来描述表示页和表示单元，《text》、《anchor》等描述的是表示元素，在 UWE 中一个可视化的页面作为一个表示页，能由不同的表示单元组成；表示单元是一个页面的逻辑部分，表示在导航设计中得到的结点；表示元素是构成一个页面的最基本元素，表示一个结点的信息，如文本、图片等。

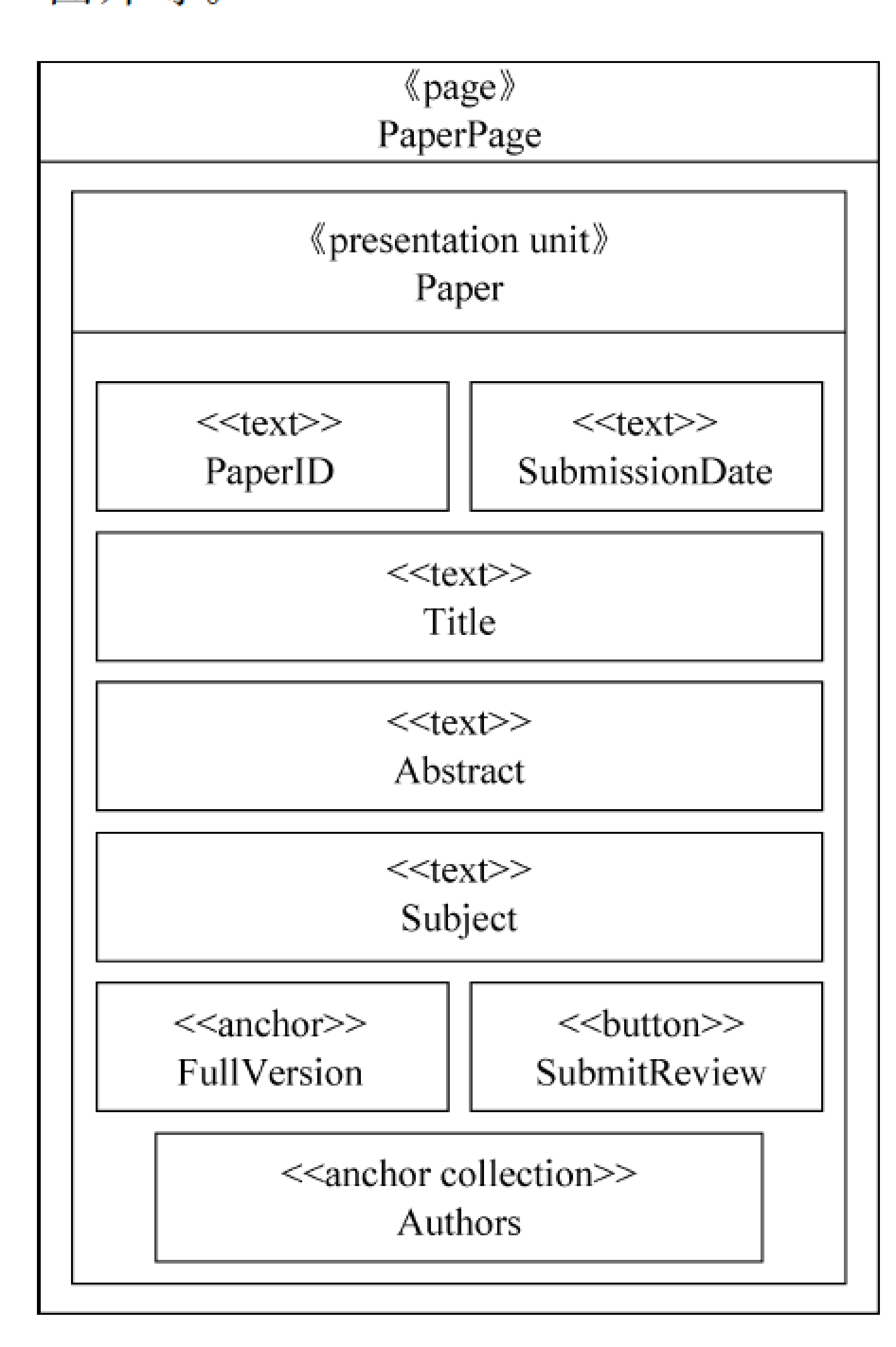


图 17-21 评审系统的论文页面表示

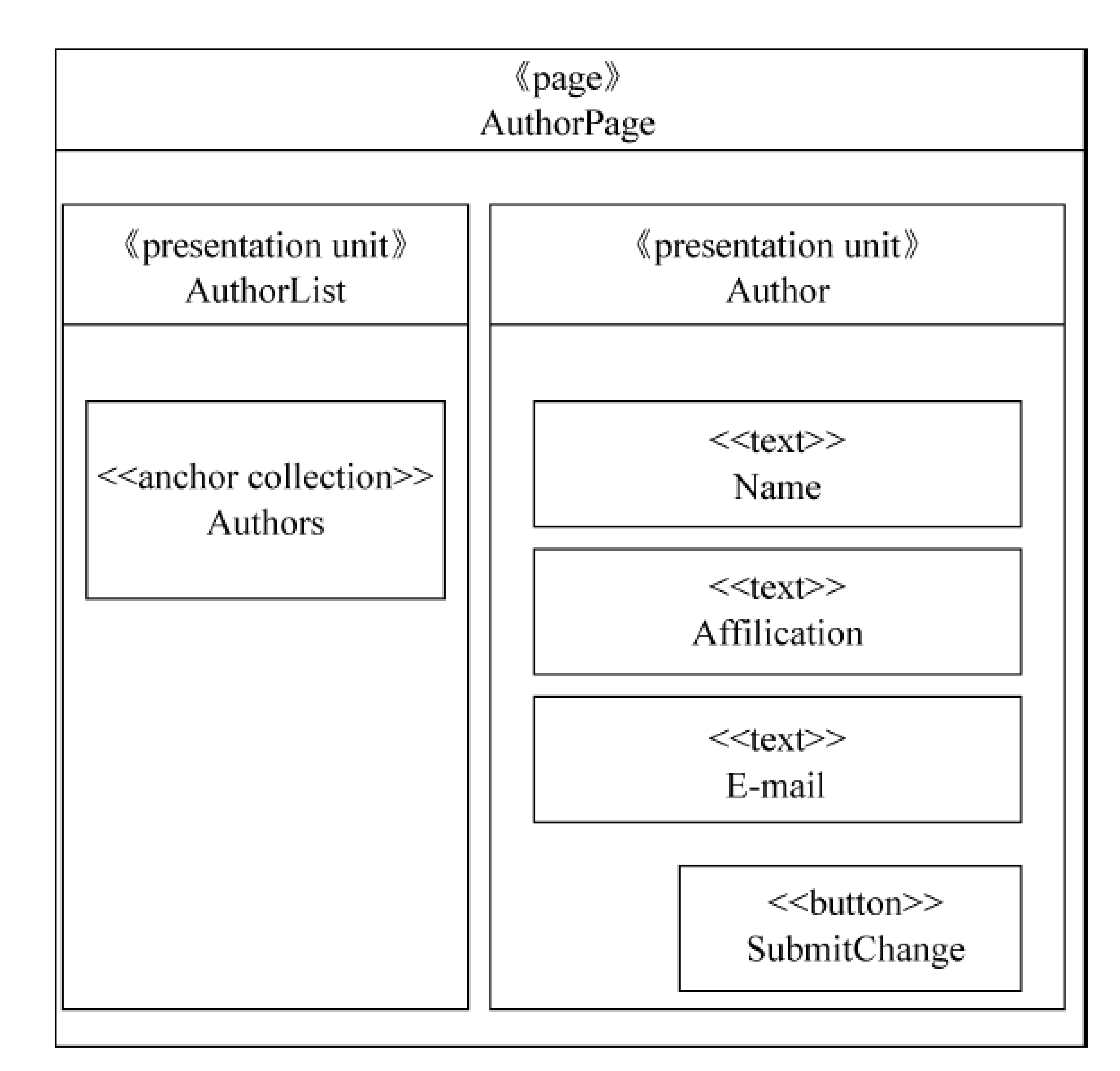


图 17-22 评审系统的作者页面表示

用户界面的行为方面，例如主席浏览一篇录用论文，能利用行为图来建模。如图 17-23～图 17-25 所示，主席在会议首页的导航栏上激活到录用论文列表的导航，从录用论文的列表中选择一篇论文，就能导航到选择的这篇论文，显示其细节内容。



图 17-23 显示选定录用论文的活动图

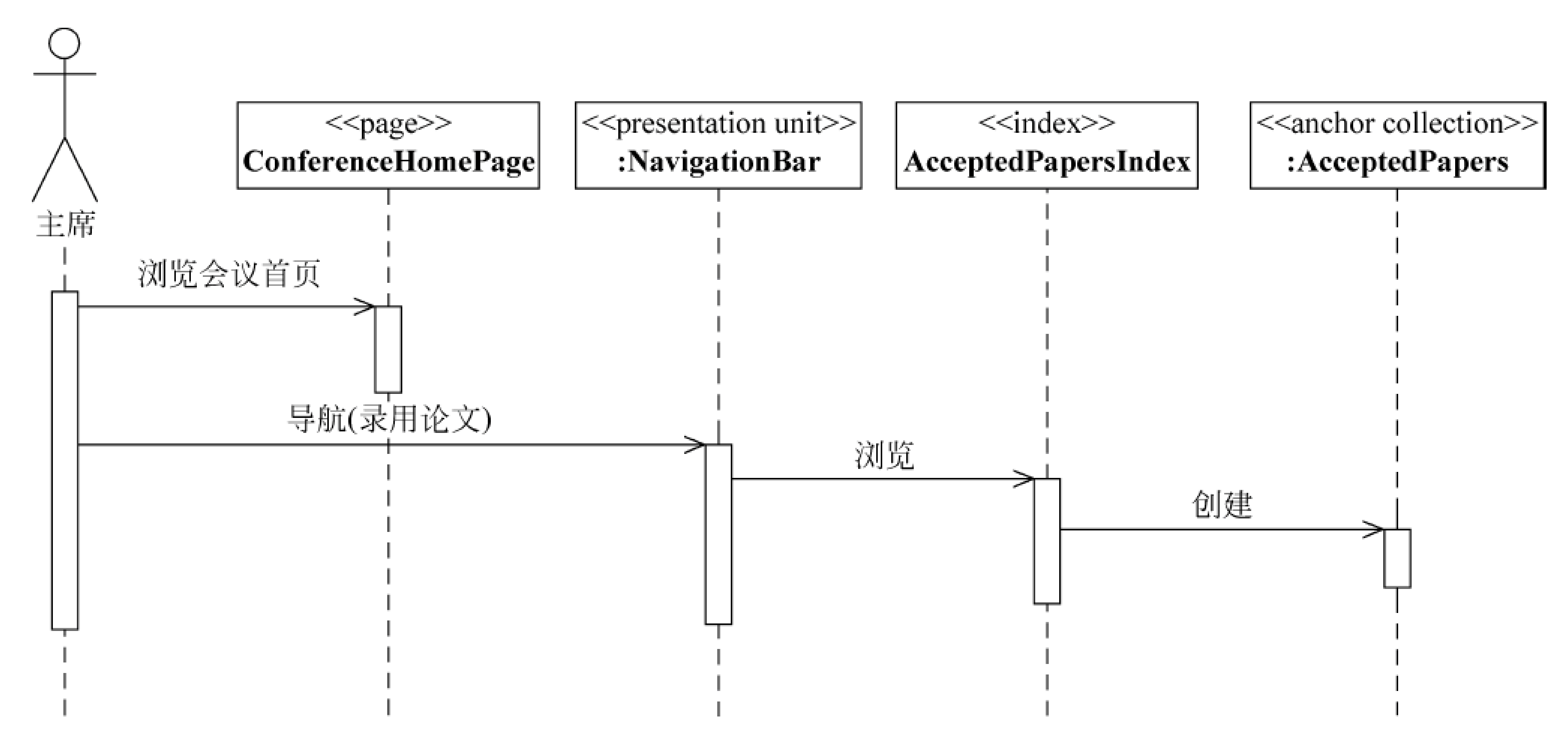


图 17-24　获取录用论文列表的时序图

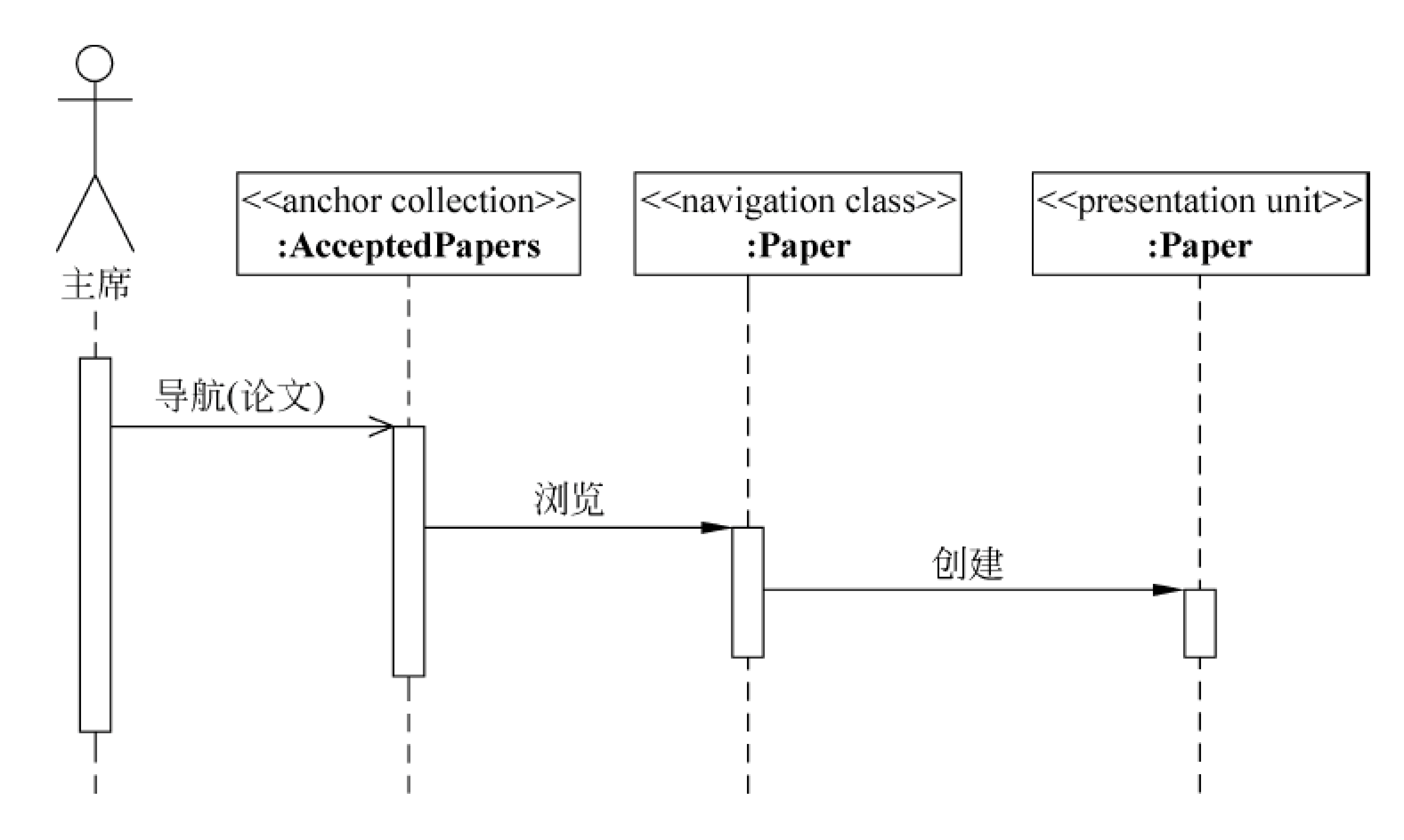


图 17-25　显示选定论文的时序图

5. 构件设计

和早期的 Web 站点相比，现代 Web 应用系统提供了更加成熟的处理功能，这些功能能够执行本地化的处理，从而动态地产生内容和导航能力；提供了适应于 Web 应用的业务领域的计算或数据处理；提供了高级的数据查询和访问；建立了与外部系统的数据接口。为了实现这些能力，必须设计和创建程序构件，这些构件在形式上与传统软件构件相同，其设计方法与传统构件相似，在此不再描述。

17.5.3　测试①

在 Web 工程过程中，Web 应用系统的测试、确认和验收是一项重要而富有挑战性的工作。Web 应用系统的测试与传统的软件测试不同，它不但需要检查和验证是否按照设计的

① 张友生. Web 工程实践研究[J]. 小型微型计算机系统，2004，25(9)：1607～1611

要求运行，而且还要测试系统在不同用户的浏览器端的显示是否合适。重要的是，还要从最终用户的角度进行安全性和可用性测试。因此必须为测试和评估复杂的Web应用系统研究新的方法和技术。

Web测试分为6部分：功能测试、性能测试、用户界面测试、兼容性测试、安全测试及接口测试。

1. 功能测试

(1) 链接测试

链接是Web应用系统的一个主要特征，它是在页面之间切换和指导用户去一些未知地址的页面的主要手段。链接测试可分为3个方面，首先测试所有链接是否按指示的那样确实链接到了该链接的页面；其次测试所链接的页面是否存在；最后保证Web应用系统上没有孤立的页面，孤立页面是指没有链接指向该页面，只有知道正确的URL地址才能访问。链接测试可以自动进行，现在已经有许多工具可以采用。链接测试必须在集成测试阶段完成，也就是说在整个Web应用系统的所有页面开发完成之后进行链接测试。

(2) 表单测试

当用户通过表单提交信息时，都希望表单能正常工作。当用户使用表单进行用户注册、登录、信息提交等操作时，必须测试提交操作的完整性，以校验提交给服务器的信息的正确性。例如，用户填写的出生日期与职业是否恰当，填写的所属省份与所在城市是否匹配等。如果使用了默认值，还要检验默认值的正确性。如果表单只能接受指定的某些值，则也要进行测试。例如，只能接受某些字符，测试时可以跳过这些字符，看系统是否会报错。

(3) 数据校验

如果根据业务规则需要对用户输入进行校验，需要保证这些校验功能正常工作。例如，省份的字段可以用一个有效列表进行校验。在这种情况下，需要验证列表完整而且程序正确调用了该列表(例如，在列表中添加一个测试值，确定系统能够接受这个测试值)。

(4) Cookies测试

Cookies通常用来存储用户信息和用户在某应用系统的操作，当一个用户使用Cookies访问了某一个应用系统时，Web服务器将发送关于用户的信息，把该信息以Cookies的形式存储在客户端计算机上，这可用来创建动态和自定义页面或者存储登录等信息。

如果Web应用系统使用了Cookies，就必须检查Cookies是否能正常工作。测试的内容可包括Cookies是否起作用，是否按预定的时间进行保存，刷新对Cookies有什么影响等。

(5) 数据库测试

在Web应用系统中，数据库起着重要的作用，数据库为Web应用系统的管理、运行、查询和实现用户对数据存储的请求等提供空间。在Web应用中，最常用的数据库类型是关系型数据库，可以使用结构化查询语言(Structured Query Language，SQL)对信息进行处理。

在使用了数据库的Web应用系统中，一般情况下，可能发生数据一致性错误和输出错误。数据一致性错误主要是由于用户提交的表单信息不正确而造成的，而输出错误主要是由于网络速度或程序设计问题等引起的，针对这两种情况可分别进行测试。

(6) 应用程序特定的功能需求

测试人员需要对应用程序特定的功能需求进行验证。尝试不同用户可以进行的所有操

作：可以提交论文、在截稿日期前修改论文、修改个人信息等。

（7）设计语言测试

Web 设计语言版本的差异可能引起客户端或服务器端严重的问题，如使用哪种版本的 HTML 等。这个问题在分布式开发环境中显得尤为重要。除了 HTML 的版本问题外，不同的脚本语言，如 Java、JavaScript、ActiveX、VBScript 或 Perl 等也要进行验证。

2. 性能测试

（1）连接速度测试

用户连接到 Web 应用系统的速度根据上网方式的变化而变化，他们或许是电话拨号，或是宽带上网。当下载一个程序时，用户可以等较长的时间，但如果仅仅访问一个页面就不会这样。如果 Web 系统响应时间太长（例如超过 5 秒钟），用户就会因没有耐心等待而离开。

（2）负载测试

负载测试是为了测量 Web 应用系统在某一负载级别上的性能，以保证 Web 系统在需求范围内能正常工作。负载级别可以是某个时刻同时访问 Web 应用系统的用户数量，也可以是在线数据处理的数量。例如，Web 应用系统能允许多少个用户同时在线？如果超过了这个数量，会出现什么现象？Web 应用系统能否处理大量用户对同一个页面的请求？

负载测试应该安排在 Web 系统发布以后，在实际的网络环境中进行测试。因为一个企业的内部员工，特别是项目组人员总是有限的，而一个 Web 应用系统能同时处理的请求数量将远远超出这个限度，所以，只有放在 Internet 上，接受负载测试，其结果才是正确可信的。

（3）压力测试

压力测试是测试系统的限制和故障恢复能力，也就是测试 Web 应用系统会不会崩溃，在什么情况下会崩溃。黑客常常提供错误的数据负载，直到 Web 应用系统崩溃，接着当系统重新启动时获得存取权。压力测试的区域包括表单、登录和其他信息传输页面等。

3. 用户界面测试

（1）导航测试

导航描述了用户在一个页面内操作的方式，在不同的用户接口控制之间，如按钮、对话框、列表和窗口等；或在不同的连接页面之间。通过考虑下列问题，可以决定一个 Web 应用系统是否易于导航：导航是否直观？Web 系统的主要部分是否可通过主页到达？Web 系统是否需要站点地图、搜索引擎或其他的导航帮助？

在一个页面上放太多的信息往往起到与预期相反的效果。Web 应用系统的用户趋向于目的驱动，很快地扫描一个 Web 应用系统，看是否有满足自己需要的信息，如果没有，就会很快地离开。很少有用户愿意花时间去熟悉 Web 应用系统的结构，因此 Web 应用系统导航帮助要尽可能准确。

导航的另一个重要方面是 Web 应用系统的页面结构、导航、菜单、连接的风格是否一致。确保用户凭直觉就知道 Web 应用系统中是否还有内容，内容在什么地方。

（2）图形测试

在 Web 应用系统中，适当的图片和动画既能起到广告宣传的作用，又能起到美化页面

的功能。一个 Web 应用系统的图形可以包括图片、动画、边框、颜色、字体、背景、按钮等。图形测试的内容如下。

① 要确保图形有明确的用途，图片或动画不要胡乱地堆在一起，以免浪费传输时间。Web 应用系统的图片尺寸要尽量小，并且要能清楚地说明某件事情，一般都链接到某个具体的页面。

② 验证所有页面字体的风格是否一致。

③ 背景颜色应该与字体颜色和前景颜色相搭配。

④ 图片的大小和质量也是一个很重要的因素，一般采用 JPG 或 GIF 压缩，最好能使图片的大小减小到 30kB 以下。

⑤ 文字回绕是否正确。如果说明文字指向右边的图片，应该确保该图片出现在右边。不要因为使用图片而使窗口和段落排列古怪或者出现孤行。

(3) 内容测试

内容测试用来检验 Web 应用系统提供信息的正确性、准确性和相关性。信息的正确性是指信息是可靠的还是误传的；信息的准确性是指是否有语法或拼写错误；信息的相关性是指是否在当前页面可以找到与当前浏览信息相关的信息列表或入口。

(4) 表格测试

需要验证表格是否设置正确。用户是否需要向右滚动页面才能看见会议召开的地点？每一栏的宽度是否足够宽，表格里的文字是否都有折行？是否有因为某一格的内容太多，而将整行的内容拉长？

(5) 整体界面测试

整体界面是指整个 Web 应用系统的页面结构设计，是给用户的一个整体感。例如，当用户浏览 Web 应用系统时是否感到舒适，是否凭直觉就知道要找的信息在什么地方？整个 Web 应用系统的设计风格是否一致？

对整体界面的测试过程，其实是一个对最终用户进行调查的过程。一般 Web 应用系统采取在主页上做一个调查问卷的形式，来得到最终用户的反馈信息。

对所有的用户界面测试来说，都需要有外部人员（与 Web 应用系统开发没有联系或联系很少的人员）的参与，最好是最终用户的参与。

4. 兼容性测试

(1) 平台测试

市场上有很多不同的操作系统类型，最常见的有 Windows、UNIX、Linux 等。Web 应用系统的最终用户究竟使用哪一种操作系统，取决于用户系统的配置，这样就可能会发生兼容性问题。同一个应用可能在某些操作系统下能正常运行，但在另外的操作系统下可能会运行失败。因此在 Web 系统发布之前，需要在各种操作系统下对 Web 系统进行平台测试。

(2) 浏览器测试

浏览器是 Web 客户端最核心的构件，来自不同厂商的浏览器对 Java、JavaScript、ActiveX 等有不同的支持。例如，ActiveX 是 Microsoft 公司的产品，是为 Internet Explorer 而设计的，JavaScript 是 Netscape 公司的产品，Java 是 Sun 公司的产品等。另外框架和层次结构风格在不同的浏览器中也有不同的显示，甚至根本不显示。不同的浏览器对安全性

和 Java 的设置也不一样。

(3) 分辨率测试

页面版式在 640×400、600×800 或 1024×768 的分辨率模式下是否显示正常？字体是否太小以至于无法浏览？或者是太大？文本和图片是否对齐？

5. 安全测试

(1) 目录设置

Web 安全的第一步就是正确设置目录。每个目录下应该有 index.html 或 main.html 页面，这样就不会显示该目录下的所有内容。

(2) 登录

有些站点需要用户进行登录，以验证他们的身份。这样对用户是方便的，他们不需要每次都输入个人资料。需要验证系统是否能阻止非法的用户名/口令登录，而能够通过有效登录？用户登录是否有次数限制？是否限制从某些 IP 地址登录？如果允许登录失败的次数为 3，在第 3 次登录时输入正确的用户名和口令，能通过验证吗？口令选择有规则限制吗？是否可以不登录而直接浏览某个页面？

Web 应用系统是否有超时的限制，也就是说，用户登录后在一定时间内(例如 30 分钟)没有点击任何页面，是否需要重新登录才能正常使用。

(3) 日志文件

在后台，要注意验证服务器日志工作正常。日志是否记录所有的事务处理？是否记录失败的注册企图？是否记录被盗信用卡的使用？是否在每次事务完成时都进行保存？记录 IP 地址吗？记录用户名吗？

(4) 脚本语言

脚本语言是常见的安全隐患，每种语言的细节有所不同。有些脚本允许访问根目录，其他只允许访问邮件服务器，但是经验丰富的黑客可以将服务器用户名和口令发送给他们自己，找出站点使用了哪些脚本语言，并研究该语言的缺陷。还需要测试没有经过授权，就不能在服务器端放置和编辑脚本的问题。

6. 接口测试

在很多情况下，Web 站点不是孤立的，可能会与外部服务器通信，请求数据、验证数据或提交订单。

(1) 服务器接口

第一个需要测试的接口是浏览器与服务器的接口。测试人员提交事务，然后查看服务器记录，并验证在浏览器上看到的正好是服务器上发生的。测试人员还可以查询数据库，确认事务数据已正确保存。这种测试可以归到功能测试中的表单测试和数据校验测试中。

(2) 外部接口

有些 Web 系统有外部接口。例如，网上商店可能要实时验证信用卡数据以减少欺诈行为的发生。测试时，要使用 Web 接口发送一些事务数据，分别对有效信用卡、无效信用卡和被盗信用卡进行验证。通常，测试人员需要确认软件能够处理外部服务器返回的所有可能的消息。

(3) 错误处理

最容易被测试人员忽略的地方是接口错误处理。通常人们试图确认系统能够处理所有错误,但却无法预期系统所有可能的错误。尝试在处理过程中中断事务,看看会发生什么情况?论文是否提交完成?尝试中断用户到服务器的网络连接,尝试中断 Web 服务器到信用卡验证服务器的连接,在这些情况下,系统能否正确处理这些错误?是否已对信用卡进行收费?

17.5.4 发布、维护

Web 发布阶段主要是把开发完成、经过初步测试的 Web 应用系统传送到 Web 站点上,供用户浏览和使用。与传统的软件系统不一样,Web 系统是需要经常更新的。这种更新包括细微的变化到大规模的变化,可以是页面内容的刷新,也可以是整个页面结构框架的更新。正是因为这种改变是经常存在的,所以大型 Web 应用系统的管理是一项艰巨的任务。对每一种变化,无论大小,都需要以一种合理的,有控制的方式进行处理。我们可把经实践证明了的软件配置管理的概念、原理和方法用到 Web 管理中。

本 章 小 结

各种 Web 应用将人们带入了信息时代,Web 应用已经成为日常生活中不可缺少的部分。实际使用的 Web 应用系统不仅在数量上不断增加,系统规模和复杂程度也在不断提高,而目前大多数的应用开发和管理实践在很大程度上依赖于开发人员个人的知识和经验,使 Web 应用系统越来越显得难以开发、管理和维护。因此迫切需要一套适用于 Web 应用系统的方法论来对相应的开发、发布和评估进行指导。本章概括了 Web 应用的发展及特点,简述了 Web 工程的概念、技术和层次,Web 工程不是软件工程的完全克隆,但是它借用了软件工程的许多基本概念和原理,强调了相同的技术和管理活动,并结合一个具体的实例说明了 Web 应用系统的分析和设计过程。

思考与练习

1. 使用类似于图 17-11 所示的图,为下面的网站建立用户层次。

(1) 图书馆管理系统。

(2) 网上银行。

2. 为第 1 题中的用户层次中的某一类用户建立用例模型。

3. 对 MVC 体系结构做一些研究,试分析是否可以进一步改进其结构。

4. 对第 1 题中的用户层次中的某一类用户设计导航模型,并给出详细的描述。

5. 查阅有关资料并做一些研究,试分析 UWE 过程有什么缺陷。

6. 对于 UWE,有开源工具 ArgoUWE 支持其开发过程,到相关网站上下载安装,为第 1 题中描述的 2 个系统建立相关的模型。

第18章 形式化方法

形式化方法在改善需求规格说明的清晰性和精确性，以及在发现重要的和敏感的错误方面具有很大的潜力。

——Steve Easterbrook

18.1 形式化方法简介

18.1.1 形式化方法的引入

在传统的软件开发过程中，人们普遍采用非形式化的图形工具和文字符号工具，例如，数据流图 DFD、模块结构图 SC、IPO 图、结构化语言、判定表、判定树等，并按照一定的设计原则和有序步骤逐步开发出目标软件，同时手工或辅助编写有关的设计文档。采用这些方法并结合彻底的评审能够得到高质量的软件。但是，非形式化的描述中可能包含矛盾、歧义性、含糊性、不完整陈述等问题，为此，引入了形式化方法进行软件开发。

形式化方法的基本含义是借助数学的方法来研究软件工程中的有关问题。在软件开发的全过程中，凡是采用严格的数学语言、具有精确的数学语义的方法，都称为形式化方法。软件的形式化方法最早可追溯到 20 世纪 50 年代后期对程序设计语言编译技术的研究，当时 Backus 提出了巴克斯范式 BNF，作为描述程序设计语言语法的元语言；20 世纪 60 年代，面对当时的软件危机，Floyd、Hoare 和 Manna 等开展的程序正确性研究推动了形式化方法的发展；20 世纪 80 年代，在硬件设计领域，形式化方法的工业应用结果掀起了软件形式化方法的学术研究和工业应用的热潮，出现了时态逻辑方法、有穷状态并发系统的模型检验方法等。近 10 年来，形式化方法的研究及其在工业中的应用得到了长足的发展。

John Wiley 在《软件工程百科全书》中对形式化方法做了如下定义：“用于开发计算机系统的形式化方法是基于数学的用于描述系统性质的技术，这样的形式化方法提供了一个框架，人们可以在该框架中以系统的方式刻画、开发和验证系统。”形式化方法基于严格的数学，能够对对象、行为等进行简洁、准确的描述，可以以一种有组织的方式来表示系统规格中的抽象层次，还可以使用数学证明来揭示系统规格中的矛盾性和不完整性，以及展示设计和规格之间的一致性情况等。

18.1.2 形式化方法的分类

软件生命周期中的分析、实现和测试等活动，分别对应于以下 3 类活动。

(1) 形式化规格

实践表明，用户需求规格说明的质量对于软件开发过程是非常重要的，如果采用自然语言描述，虽然具有易读、易理解的优点，但是自然语言描述的规格说明具有模糊性和二义性，会给下一步的开发工作造成理解上的困难，而且自然语言是非形式化的，无法得到计算机的直接支持，使软件生成自动化几乎不可能。

程序设计语言作为计算机专用的语言，可以用来描述需求规格说明，但它着重描述的是"如何做"，而不是"做什么"的问题，因此不太适合描述抽象程度较高的需求规格说明。近年来发展的第 4 代语言 4GL，更多地采用了过程抽象和数据抽象技术，关注系统要实现的功能，而忽略实现过程，但这种语言的出发点仍在系统的设计方面，作为需求规格的描述语言仍存在不足。

因此，一种综合性的专用于需求规格说明的形式规格说明语言应运而生。这种形式规格说明语言克服了自然语言和程序设计语言的不足，应用形式化、规范化的数学理论，严格定义系统"做什么"的形式语义模型，并支持自动程序转换系统将需求规格说明的语义模型转化为可执行代码。

根据对需求规格说明的定义方式，形式化规格方法可以分为以下两大类。

① 面向模型的形式化方法

面向模型的形式化方法又称为基于状态描述的形式化方法，其基本思想是利用域、元组、集合、序列、映射、包等已知特性的数学抽象概念来为目标软件系统的状态特征和行为特征构造形式语义模型，语义模型就作为需求规格的形式说明。主要代表有 VDM 方法(维也纳方法)、Z 方法等。

② 代数构造形式化方法

代数形式化方法为需求规格说明提供一些特殊的构造机制，并以代数构造方式描述目标系统的结构、功能，如进程代数等。

(2) 形式化验证

众所周知，软件开发中的很大一部分错误是在需求分析和规格说明的早期阶段引入的，并且随着开发的深入而逐渐放大。而且，错误发现得越晚，修改错误的代价也就越大。在传统的软件开发方法中，除了各阶段评审发现的错误外，更多的错误是到编码结束后的测试阶段才被检测出来的。在形式化方法中，在用形式化规格完成后进行形式化验证，可以提前发现错误，同时修改错误的代价也减小了。

形式化验证的方法主要有模型检验和定理证明。模型检验是一种基于有限状态模型并检验该模型特性的技术，主要适用于有穷状态系统，优点是可以完全自动化并且验证速度快。定理证明采用逻辑公式来规格系统及其性质，逻辑是由具有公理和推理规则的形式化系统给出，应用这些公理或推理规则来证明系统具有某些性质，可以处理无限状态空间问题，但是定理证明的实施需要定理证明器的支持。

(3) 程序求精

程序求精，又称为程序变换，是将自动推理和形式化方法相结合而形成的一门新技术，

研究从抽象的形式化规格推演出具体的面向计算机的程序代码的全过程。程序求精的基本思想是用一个抽象程度低、过程性强的程序代替一个抽象程度高、过程性弱的程序，并保持它们之间功能的一致性。因此，程序开发过程实际上就是从最高层的程序开始，通过一系列的细化步骤，每一步都降低一些抽象程度或增加一些可执行性，最终得到能够指导计算机明确执行的程序代码。在细化的过程中要保证程序的正确性，可以通过一系列规则来保证，也可以在事后采用验证工具来证明。

本章着重介绍形式化规格方法。

18.1.3　形式化规格方法的主要思想

形式化规格方法的主要思想是利用形式化规格说明语言严格地定义用户需求，并采用数学推演的方法证明需求定义的性质，如一致性、公平性等。对于复杂问题，可能无法验证整个需求定义的完整性，但仍可以为避免某些要点的疏漏而建立数学断言，予以形式证明或反驳。从这个意义上讲，形式化方法是克服需求分析阶段中主要困难(不精确性、不一致性和不完全性)的有效途径。

形式化规格说明语言主要包括严格的语法定义、严格的语义定义以及一系列的数学推演规则。这些规则不仅说明某些数学性质在软件规格说明中是否成立，也说明了软件实现与软件规格说明之间的满足关系。

规格说明的语法一般基于集合论、数理逻辑或代数学，有的还包括程序设计语言中的控制流结构，如顺序、条件、循环等。

规格说明语言的语义是其所有语法符号的意义的数学描述。经典的语义定义方法包括指称语义、代数语义和操作语义方法。指称语义首先用数学的方法确定规格说明语言的语义域，然后将其所有语法成分映射为语义域中的对象或语义域上的函数；代数语义则将规格说明中的某些结构化构件解释为多类代数，通过代数工具，如范畴论研究规格说明的代数性质、模块组装运算以及软件设计相对于规格说明的实现关系；操作语义首先定义抽象机，然后将规格说明的语义解释为抽象机的动作序列。

推演规则与形式化规格说明语言的数学基础和语义定义方法密切相关。例如，以集合论和谓词逻辑为基础的Z语言就包含了原数学系统中有关的规则。规则必须在规格说明语言的语义系统中可证。因此，可以认为规则是派生的语义定义，可以直接应用于规格说明的性质证明并简化推演过程。

下面将讨论两种形式化方法。

18.2　Petri 网

18.2.1　基本概念

Petri 网最早是由 Carl Adam Petri 于 1962 年在他的博士论文《自动机通信》中提出来的。任何系统都可抽象为状态、活动(或事件)及其之间关系的三元结构，Petri 网用位置表示状态，转换表示活动。转换即改变状态，位置决定能否产生转换，转换与位置的依赖关系

用流来表示①。

Petri 网的静态结构表示成三元组 $N=(P,T,F)$，其中

① $P=\{p_1,p_2,\cdots,p_n\}$是有穷位置集合。

② $T=\{t_1,t_2,\cdots,t_n\}$是有穷转换集合，且 $P\cup T\neq\Phi, P\cap T\neq\Phi$。

③ $F\subseteq(P\times T)\cup(T\times P)$为由一个 P 元素和一个 T 元素组成的有序偶集合。

在图形表示中，用圆圈表示位置，用黑短线表示转换，用有向弧表示位置与转换的有序偶。例如，对于 $P=\{p_1,p_2,p_3,p_4,p_5,p_6\}$、$T=\{t_1,t_2,t_3,t_4,t_5\}$和 $F=\{(p_1,t_1),(t_1,p_2),(t_1,p_3),(p_2,t_2),(p_3,t_3),(t_2,p_4),(t_3,p_5),(p_4,t_4),(p_5,t_4),(t_4,p_6),(p_6,t_5),(t_5,p_1)\}$的 Petri 网结构 $N=(P,T,F)$，其图形表示如图 18-1 所示。

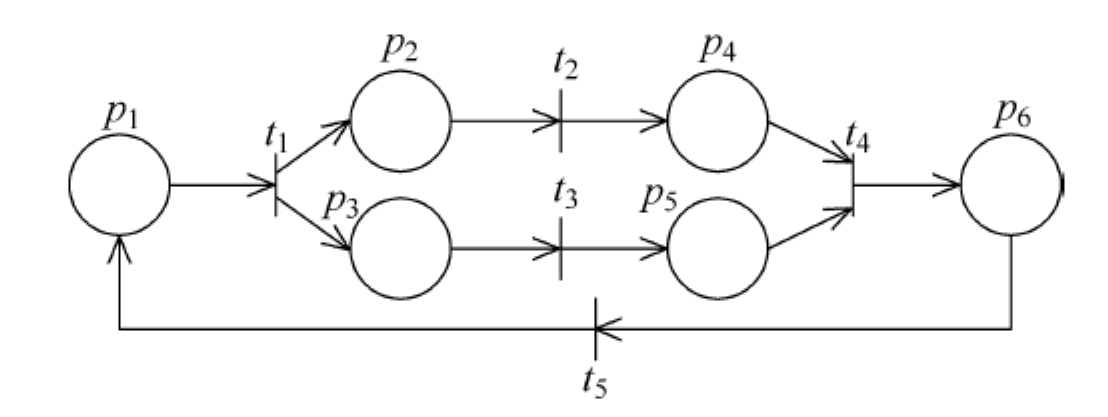


图 18-1　Petri 网结构的图形表示

前集和后集：对于一个 Petri 网结构 $N=(P,T,F)$，设 $x\in(P\cup T)$，令

$$*x=\{y\mid\exists y:(y,x)\in F\}$$

$$x*=\{y\mid\exists y:(x,y)\in F\}$$

称 $*x$ 为 x 的前集或输入集，$x*$ 为 x 的后集或输出集。

Petri 网除了具有上述的静态结构外，还包括描述系统动态行为的机制，允许位置中包含令牌，令牌可以根据转换重新分布。具有动态特征的 Petri 网定义如下。

位置/转换 Petri 网：简称为 Petri 网，形式上定义为一个六元组 $PN=(P,T,F,K,W,M_0)=(N,K,W,M_0)$，其中，

$N=(P,T,F)$是一个 Petri 网结构。

$K: P\rightarrow Z^+\cup\{\infty\}$是位置的容量函数($Z^+$是正整数集合)，规定位置中可以包含的令牌的最大数目。对于任一位置 $p\in P$，$K(p)$表示向量 K 中位置 p 所对应的分量，用于表示位置 p 的容量。

$W: F\rightarrow Z^+$是弧集合上的权函数，规定了令牌传递的加权系数，对于任一弧 $f\in F$，以 $W(f)$表示向量 W 中弧 f 所对应的分量，用于表示弧 f 上的加权系数。

$M: P\rightarrow Z$(非负整数集合)是位置集合上的标识向量，对于任一位置 $p\in P$，$M(p)$表示标识向量 M 中位置 p 所对应的分量，称为位置 p 上的标识或令牌数目，并且必须满足 $M(p)\leqslant K(p)$，M_0 是初始标识向量。

在 Petri 网的图形表示中，弧(x,y)上的 W 值标注在弧(x,y)上，当 $W(f)=1$ 时，省略标注；$K(p)$写在位置 p 的圆圈旁边，当 $K(p)=\infty$时，省略 $K(p)$的标注；$M(p)$的值用实心点或者数字表示，在位置 p 对应的圆圈中标注 $M(p)$个实心点，每个实心点称为一个标志或令牌。对应于图 18-2 所示的 Petri 网，$M_0=(1,0,0,0,0,0)$，$W(t_6,p_1)=W(t_3,p_5)=$

① 古天龙. 软件开发的形式化方法. 北京：高等教育出版社，2005

$W(t_4,p_1)=2, W(t_2,p_4)=3, W(p_1,t_1)=W(t_1,p_2)=W(p_2,t_2)=W(p_4,t_4)=W(t_1,p_3)=W(p_3,t_3)=W(p_5,t_4)=W(p_1,t_5)=W(t_5,p_6)=W(p_6,t_6)=1$。

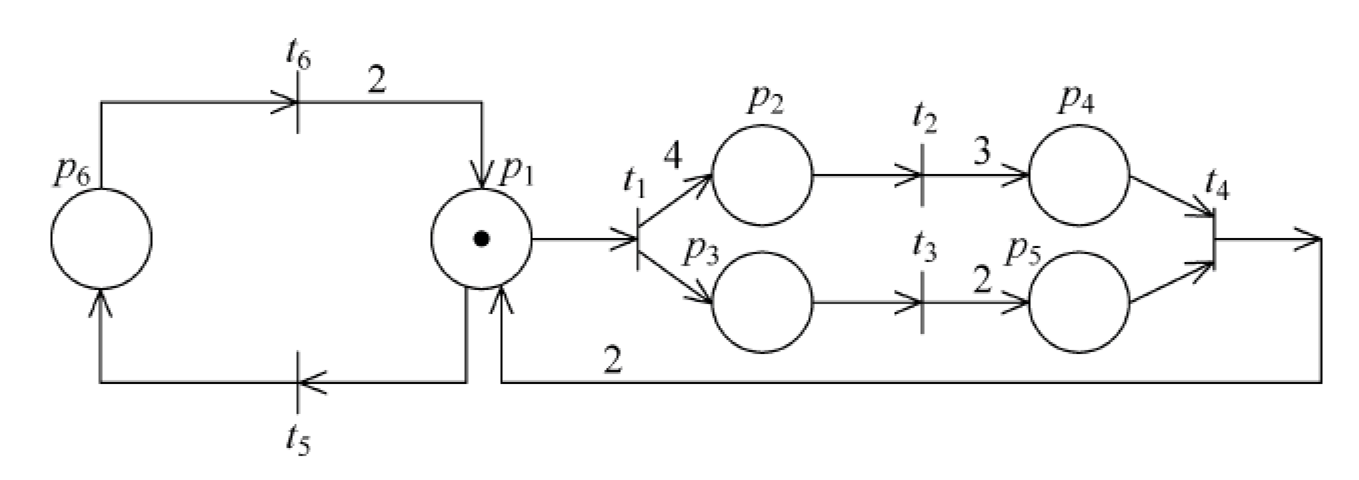


图 18-2 一个具有 M_0 和 W 的 Petri 网

转换发生规则：Petri 网的动态行为是通过转换引起标识改变来体现的，下面是发生转换的条件和规则①。

(1) 转换 t 可发生的条件

对于 Petri 网 $PN=(P,T,F,K,W,M)$，若在标识 M 下，如果对于 $\forall p_1$，都有 $p_1 \in {}^*t \Rightarrow M(p_1) \geqslant W(p_1,t)$，且对 $\forall p_2$，都有 $p_2 \in t^* \Rightarrow K(p_2) \geqslant M(p_2)+W(t,p_2)$，则称 t 在 M 下可发生，记为 $M[t>$。

(2) 转换 t 发生的结果

对于 Petri 网 $PN=(P,T,F,K,W,M)$，若 t 在 M 下可发生，发生后将 M 变成新标识 M'，记为 $M[t>M'$ 或 $M \xrightarrow{t} M'$，并称 M' 为 M 的后继标识。对 $\forall p \in P$，$M'(p)$ 可通过式(18-1)进行计算：

$$M'(p)=\begin{cases} M(p)-W(p,t) & \text{当 } p \in {}^*t - t^* \\ M(p)+W(t,p) & \text{当 } p \in t^* - {}^*t \\ M(p)-W(p,t)+W(t,p) & \text{当 } p \in t^* \cap {}^*t \\ M(p) & \text{当 } p \notin t^* \cup {}^*t \end{cases} \tag{18-1}$$

没有任何输入位置的转换称为源转换，一个源转换是无条件的，源转换的发生只会产生标识，而不消耗任何标识；一个没有任何输出位置的转换称为阱转换，阱转换的发生将消耗标识，而不产生任何标识。

Petri 网具有丰富的结构描述能力，下面以图 18-3 为例给出几种结构关系。

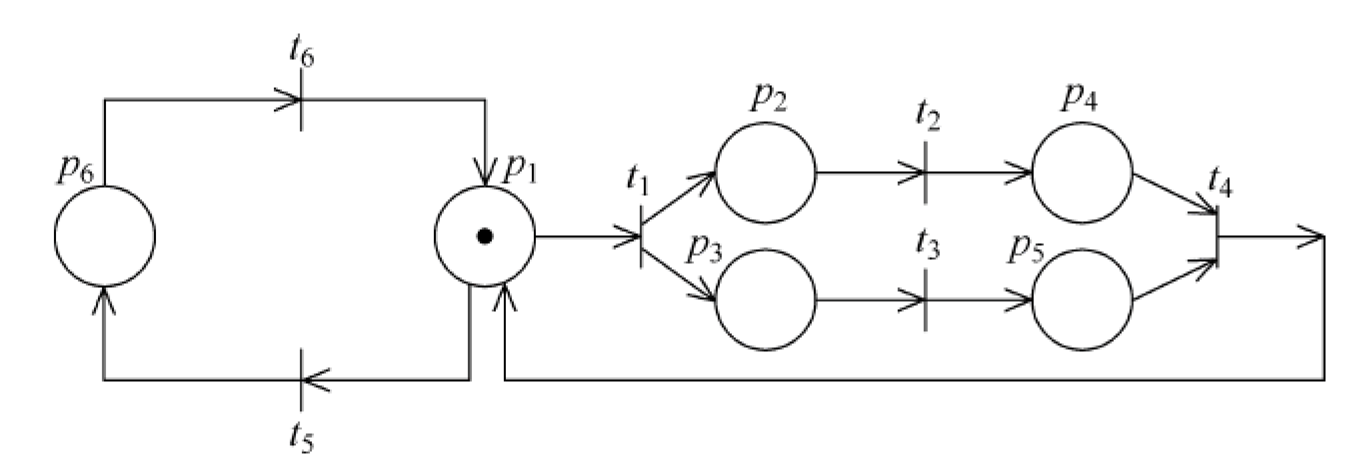


图 18-3 一个 Petri 网

(1) 顺序关系

设 M 为 Petri 网 PN 的一个标识，若存在 t_1 和 t_2，使 $M[t_1>M'$，且 $\neg M[t_2>$，$M'[t_2>$，即

① 杨文龙，古天龙. 软件工程. 北京：电子工业出版社，2007

在 M 标识下，t_1 可发生，t_2 不可发生，且 t_1 的发生使 t_2 可发生，即 t_2 的发生以 t_1 的发生为条件，则称 t_1 和 t_2 在 M 下有顺序关系。如在图 18-3 中，t_1 和 t_2，t_1 和 t_3 在 M_0 下有顺序关系。

（2）冲突关系

设 M 为 Petri 网 PN 的一个标识，若存在 t_1 和 t_2，$M[t_1>$ 和 $M[t_2>$，并满足 $M[t_1>M_1 \Rightarrow \neg M_1[t_2>$，且 $M[t_2>M_2 \Rightarrow \neg M_2[t_1>M'[t_2>$，即在 M 标识下，t_1 和 t_2 都可发生，但它们中只有一个能发生，任何一个转换的发生都会使另一个转换不可发生，则称 t_1 和 t_2 在 M 下有冲突。如在图 18-3 中，t_1 和 t_5 在 M_0 下冲突。

（3）并发关系

设 M 为 Petri 网 PN 的一个标识，若存在 t_1 和 t_2，使 $M[t_1>$ 和 $M[t_2>$，并满足 $M[t_1>M_1 \Rightarrow M_1[t_2>$，且 $M[t_2>M_2 \Rightarrow M_2[t_1>M'[t_2>$，即在 M 标识下，t_1 和 t_2 都可发生，且它们中任何一个转换的发生都不会使另一个转换不可发生，即两个转换互不影响，则称 t_1 和 t_2 在 M 下有并发关系。如在图 18-3 中，$M_0[t_1>M$，则 $M=(0,1,1,0,0,0)$，t_2 和 t_3 在 M 下并发。

（4）碰撞

如果让 p_1 和 p_2 各有一个标识，并规定位置容量均不能超过 1，这时 t_1 不能发生，因为 t_1 的发生会使 p_2 的容量超过 1，称这种现象为碰撞。

（5）混惑关系

某些情况下，一个 Petri 网中同时存在着并发和冲突，而且并发的发生可能会引起冲突的消失（减少）或出现（增加），称这种情况为“混惑”。如在图 18-4 中，t_1 和 t_3 在 $M_0=(1,1,0,1,0,0)$ 下并发，若 t_3 先于 t_1 发生，则 p_3 获得标识，此时 t_1 和 t_2 在 $M'=(1,1,1,0,0,0)$ 下冲突，若 t_2 发生，则 p_6 获得标识，系统进入终态 $M_1=(1,0,0,0,0,1)$；若 t_1 发生，则 p_5 获得标识，系统进入另一个终态 $M_2=(0,0,1,0,1,0)$。从初始状态 M_0 到终态 M_2 的另一种可能是让 t_1 先于 t_3 发生，则 t_1 和 t_2 就不会发生冲突；还有一种可能是 t_1 和 t_3 并发，也可使系统进入终态 M_2。从标识向量 $M_0 \rightarrow M_2$ 的变化中，无法判断其间是否出现过冲突，使外部环境对系统难以控制，因此，存在“混惑”的系统不是好系统。

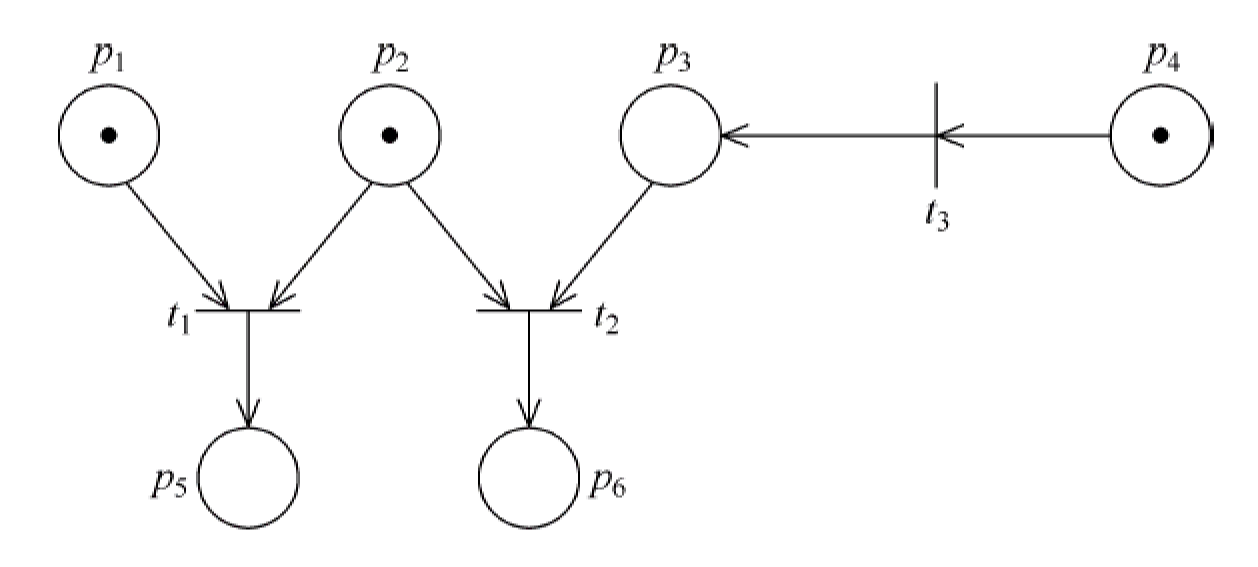


图 18-4　说明“混惑”的 Petri 网

早期所研究的 Petri 网结构，所有弧上的权都隐含为 1，所有位置的容量都隐含为无穷，习惯上把这种 Petri 网称为简单 Petri 网，而把有限容量和权值为任意正整数的 Petri 网称为 P/T 网。对于简单 Petri 网，由于容量函数 K 和权函数 W 已经没有任何限制作用，一般把这种 Petri 网记为 $PN=(N,M)=(P,T,F,M)$，这种 Petri 网的转换规则可以简化为：若 $M[t>$，当且仅当 $\forall p \in P, M(p) \geq 1$，$t$ 发生后，$M[t>M'$，并有

$$M'(p)=\begin{cases}M(p)-1 & \text{当 } p\in {}^{*}t-t^{*}\\ M(p)+1 & \text{当 } p\in t^{*}-{}^{*}t\\ M(p) & \text{其他}\end{cases}$$

可以证明，经过适当的处理，每个P/T网都可以改造为行为等效的简单Petri网。下面讨论简单Petri网的基本性质，也都适用于P/T网。

18.2.2　Petri网的性质

1. 可达性

可达性是研究任何系统动态行为的基础。按照转换发生的规则，转换的发生将改变令牌的分布(产生新的标识)。对于初始标识 M_0，如果存在一系列转换 $t_1,t_2,\cdots,t_n$ 的发生使 M_0 转换为 M_n，则称标识 M_n 是从 M_0 可达的，记为 $M_0[\sigma>M_n$，其中 $\sigma=M_0t_1M_1t_2M_2\cdots t_nM_n$，或简记为 $\sigma=t_1t_2\cdots t_n$，称为转换的发生序列。对于Petri网 $PN=(N,M_0)$，所有可达的标识组成一个可达集，记为 $R(N,M_0)$ 或 $R(M_0)$；从 M_0 出发的所有可能发生的序列组成一个发生序列集合，记为 $L(N,M_0)$ 或 $L(M_0)$。

2. 有界性和安全性

在 $PN=(N,M_0)$ 中，若存在一个非负整数 k，使 M_0 的任一可达标识的每个位置中的标记数都不超过 k，即 $\exists k\in Z^+$，对 $\forall M\in R(M_0)$，都有 $k\geqslant M(p)$，则称位置 p 为 k 有界。如果 PN 中每个位置都是 k 有界，则称 PN 为 k 有界。如图18-5(a)所示的Petri网是有界的，图18-5(b)所示的Petri网不是有界的，因为存在一个发生序列 $\sigma=t_2t_1t_2t_1\cdots$，使 p_5 中的标识无限制地增多。

如果位置 p 为1有界，则称位置 p 是安全的。如果 PN 中每个位置都是安全的，则称 PN 是安全的。

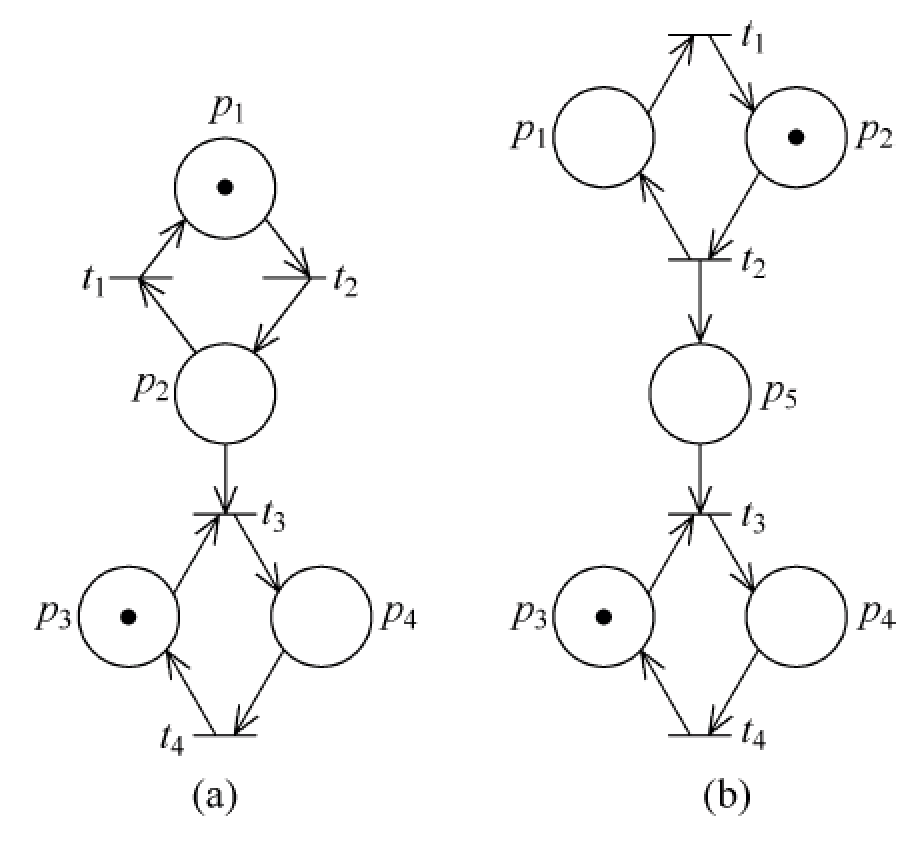


图18-5　有界和无界的Petri网

3. 活性

在操作系统中，活性是与系统安全无死锁紧密相关的一个性质。一个Petri网 (N,M_0) 被称为活的，仅当从 M_0 可达的任一标识出发，都可以通过执行某一转换序列而最终发生任一转换，即对任何 $M\in R(M_0)$，任何 $t\in L(M)$，从 M 出发，可以通过执行某一转换序列而最终引发 t。这意味着，无论选择什么样的发生序列，一个活的Petri网都可以保证无死锁操作。

活性是系统的理想特性，但这是不现实的。为此，我们放宽对活性的限制，并定义不同的活性等级。Petri网 (N,M_0) 中的一个转换 t 被称做：

① L_0-活的(死的)，仅当 t 在 $L(M_0)$ 中的任何发生序列中都无法发生。

② L_1-活的(可能发生),仅当 t 在 $L(M_0)$中的一些发生序列中至少可发生一次。

③ L_2-活的,对任一正整数 k,仅当 t 在 $L(M_0)$中的一些发生序列中至少可发生 k 次。

④ L_3-活的,仅当 t 在 $L(M_0)$中的一些发生序列中可以经常无限制地发生。

⑤ L_4-活的(活的),仅当 t 在 $R(M_0)$中的每个标识 M,是 L_1-活的。

一个 Petri 网(N,M_0)被称为 L_k-活的,仅当网中每个转换是 L_k-活的($k=0,1,2,3,4$)。L_4-活的是最强的,与前面定义的活性是一致的。很容易看出:

L_4-活的⇒L_3-活的⇒L_2-活的⇒L_1-活的

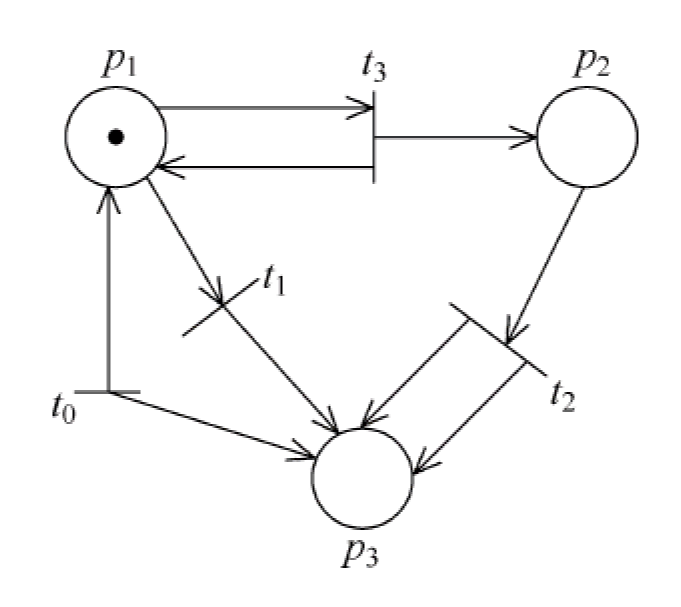


图 18-6 一个具有 $L_k(k=0,1,2,3)$-活转换的 Petri 网

在图 18-6 所示的 Petri 网中,t_0 永远无法发生,是 L_0-活的(死的);一旦 p_1 因发生 t_1 而失去标识,就无法再获得标识,即 t_1 是 L_1-活的;在 p_1 有标识的情况下,可能发生 t_3,且 p_1 不丢失标识,t_3 可能反复连续发生,因此 t_3 是 L_3-活的;t_1 发生后,t_3 不能再发生,t_2 最大的发生次数为 p_2 中的标识数,即 t_3 已经发生的次数,而 t_3 发生的次数可任意指定,所以 t_2 为 L_2-活的。

4. 可逆性

在 Petri 网 $PN=(N,M_0)$中,如果 $\forall M\in R(M_0)$,$M_0\in R(M)$,则称该 Petri 网是可逆的。对于可逆的 Petri 网,存在发生序列 $\sigma=t_1t_2\cdots t_n$,从 $\forall M\in R(M_0)$返回到 M_0。因此,一个可逆网可以返回到初始状态或初始标识。在很多应用中,仅要求系统回到某个特定状态,而无需返回到初始状态,这个特定的状态称为主状态。对于 $R(M_0)$中的每个标识 M,其主状态都是可达的。

5. 可覆盖性

在 Petri 网 $PN=(N,M_0)$中,如果对于某个标识 M,$\exists M'\in R(M_0)$,使其对于 $\forall p\in P$,有 $M'(p)\geqslant M(p)$,则称 M 是可覆盖的。

可覆盖性与 L_1-活的(潜在可发生性)紧密相关。如果 M 是转换 t 发生所必需的最小标识,那么,M 是不可覆盖的,当且仅当 t 是死的(L_0-活的)。亦即,t 是 L_1-活的,当且仅当 M 是可覆盖的。

6. 持久性

在 Petri 网 $PN=(N,M_0)$中,如果对于任意两个可发生的转换,其中一个转换的发生不会使另一个转换不发生,则称 PN 是持久的。也就是说,持久网中的一个转换一旦发生,将保持这种可发生性直到它发生为止。不存在冲突的 Petri 网具有持久性,所以所有位置都只有一条输入和一条输出弧的 Petri 网具有持久性。

7. 同步距离

对于 Petri 网 $PN=(N,M_0)$,任意两个转换 t_1、$t_2\in T$ 的同步距离定义为

$d_{12}=\max|\sigma(t_1)-\sigma(t_2)|$，其中，$\sigma$ 是从任意一个标识 $M\in R(M_0)$ 开始的发生序列，$\sigma(t_i)$ 为发生序列 σ 中 $t_i(i=1,2,\cdots)$ 发生的次数。例如，在图 18-7 所示的 Petri 网中 $d_{12}=1,d_{34}=1,d_{13}=\infty$。

同步距离用来刻画不同形式的同步关系，是两个转换之间这种相对关系的一种定量描述，是条件/事件系统中两个事件间相互独立程度的一种量度。

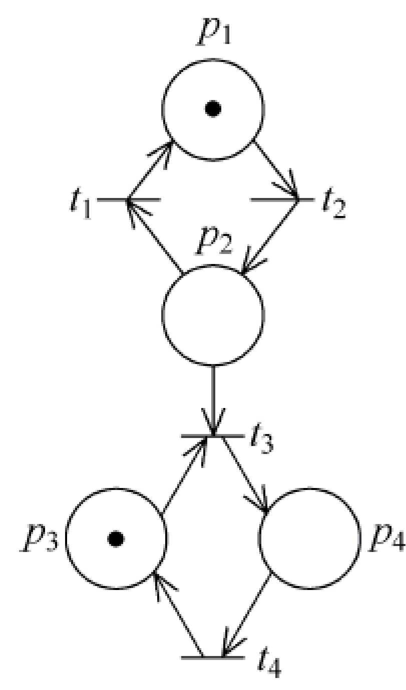


图 18-7 说明同步距离的 Petri 网

8. 公平性

这里介绍两种基本的公平性：有界公平性和无条件公平性。对于 Petri 网 $PN=(N,M_0)$ 中的两个转换 t_1 和 t_2，如果其中一个转换不发生，另一个转换可以发生的最大次数为有界的，则称这两个转换具有有界公平关系。若 PN 中任意一对转换都存在有界公平关系，则称该网为有界公平网。

对于一个发生序列 σ，若 $|\sigma|$（发生序列中转换的数目）为有限数，或 PN 中任何转换都在 σ 中无限次出现，则称 σ 为无条件公平的。若从 $R(M_0)$ 中某个 M 开始的每个发生序列都是无条件公平的，则称该 Petri 网为无条件公平网。

18.2.3 Petri 网的分析

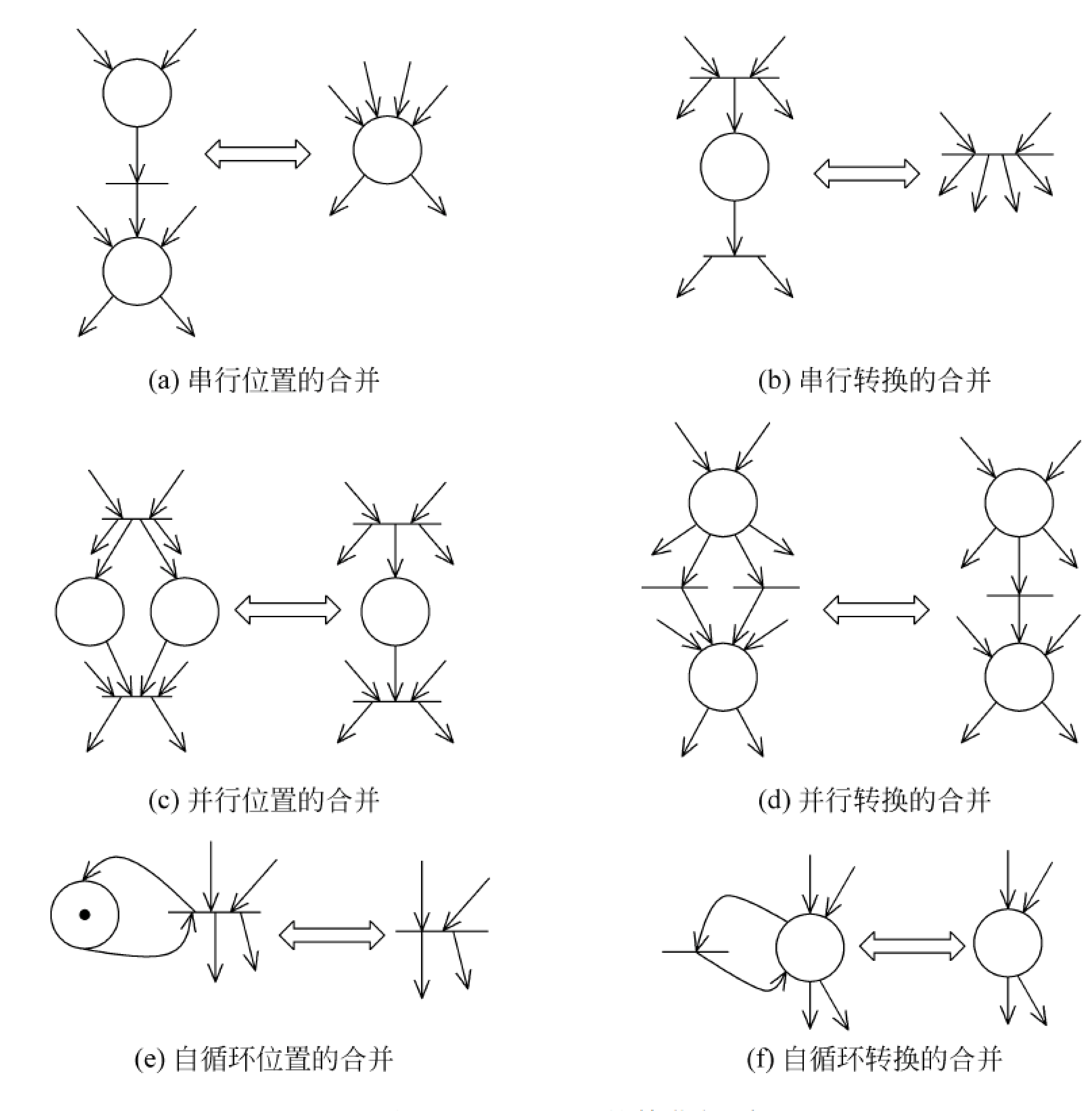
Petri 网的特性分析方法有 3 类：结构化简、可覆盖树、矩阵方程求解。第一种方法是在保证网系统要分析的性质不变的情况下进行化简，涉及到转换方法的研究；第二种方法实质上包含了所有可达标识或它们的可覆盖标识的枚举，适用于规模较小的网；第三种方法求解能力强，但在很多情况下，仅适用于 Petri 网的一些特殊子类或特殊情况。

1. 结构化简

结构化简是处理复杂问题的一种方法，其基本原则是在保持化简前、后 Petri 网所具有的某些性质不变的前提下，将多个不同的位置或转换抽象为单个的位置或转换。化简可以将冗余或等价的位置或转换删除或合并，下面是化简的几条规则。设 (N,M_0) 和 (N,M_0') 分别为化简前后的 Petri 网，运用以下化简规则，当且仅当 (N,M_0) 是活的、安全的和有界的，则 (N,M_0') 是活的、安全的和有界的。图 18-8(a)～(f) 分别列出了几个基本简化规则，它们能保持 Petri 网的活性、有界性和安全性。

图 18-9 是应用上述规则对 Petri 网进行化简的例子。

图 18-9(a) 中先发生 t_2 转换，p_1 中的标记移到 p_2，将 t_1、p_1、t_2 合并为 t_{12}（串行转换的合并），同样将 t_3、p_4、t_4 合并为 t_{34}，得到如图 18-9(b) 所示的 Petri 网。再分别将图 18-9(b) 所示的 Petri 网中自循环转换 t_{12} 和自循环转换 p_3 合并，即得到化简的最后结果，如图 18-9(c) 所示。

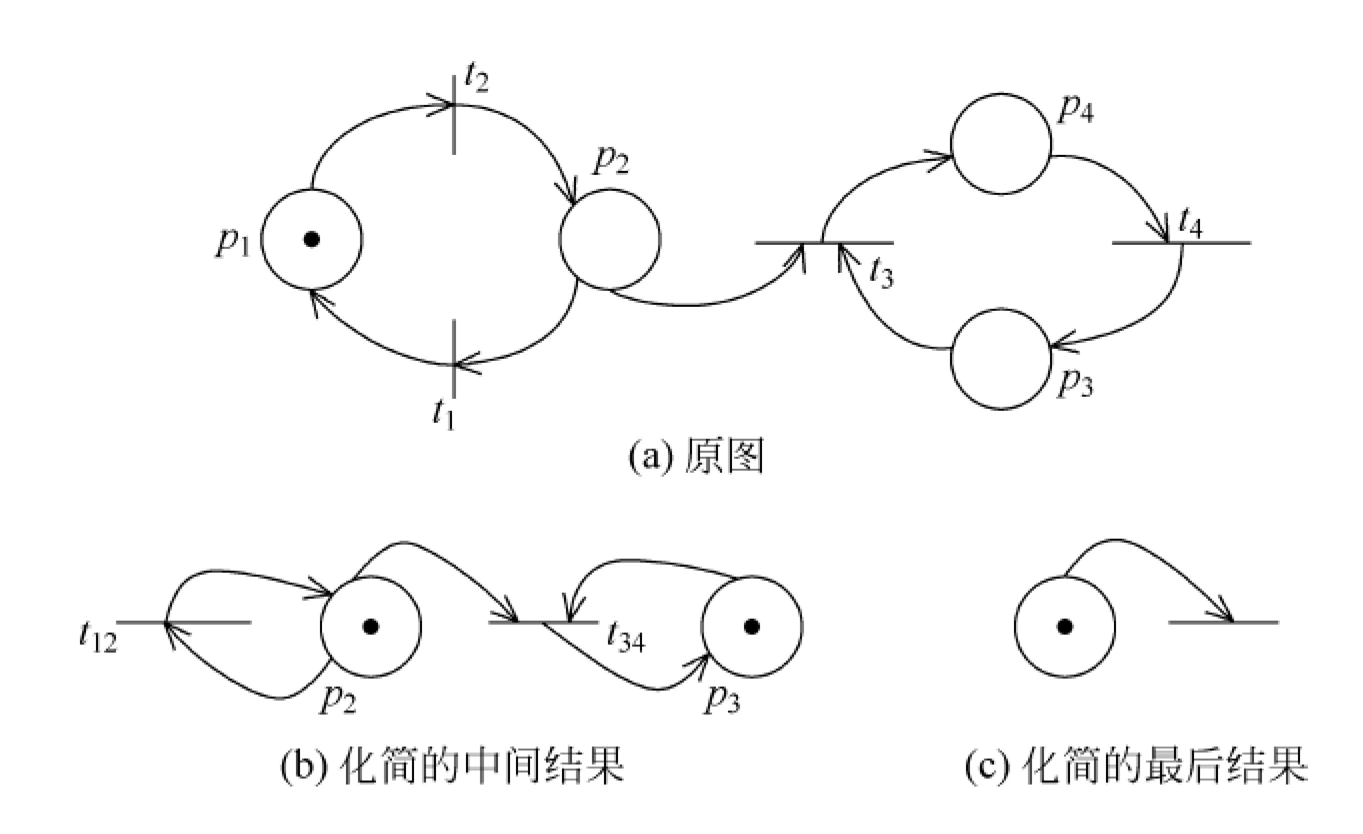


(a) 串行位置的合并　(b) 串行转换的合并

(c) 并行位置的合并　(d) 并行转换的合并

(e) 自循环位置的合并　(f) 自循环转换的合并

图 18-8　Petri 网的简化规则



(a) 原图

(b) 化简的中间结果　(c) 化简的最后结果

图 18-9　Petri 网化简的例子

2. 可覆盖树

对于一个给定的 Petri 网 $PN=(N,M_0)$，从初始的标识向量 M_0 开始，经过一系列的转换，可以得到数量与转换一样的"新的"标识，从各个新的标识开始，又可以达到更多的标识。用树结构来表示这个过程，初始标识 M_0 作为树根，由可达的标识作为树的后继结点，结点之间的弧表示标识和转换之间的关系 $M[t>M'$，称这样的树为标识树。如果 Petri 网是无

界的，所构造的标识树可能无限增长，为了使得到的标识树保持有限，引入一个特殊符号 ω，表示“无限”，ω 具有如下性质：

$$\forall k \in Z: \omega > k, \quad \omega \pm k = \omega \text{ 且 } \omega \geqslant \omega$$

引入 ω 的标识树称为可覆盖树，也称可达树。在可覆盖树中，对于结点 M，从 M_0 到 M 的路径上存在结点 M'，M' 满足

$$\forall p \in P: M(p) \geqslant M'(p) \text{ 且 } M \neq M'$$

则称 M' 是可覆盖的。对于这样的 $M'(p)$ 中每个满足 $M(p) > M'(p)$ 的 p，用 ω 重置 $M(p)$，即得到覆盖树。

覆盖树的构造算法如下。

(1) 将初始标识 M_0 作为树根，标记为 new。

(2) 若树的所有结点都为 old，算法结束。

(3) 若存在 new，重复以下步骤。

① 选择一个标记为 new 的标识 M。

② 如果从根结点到 M 的路径上有与 M 相同的标识，则将 M 标记为 old。

③ 如果 M 中没有转换可以发生，则将 M 标记为 dead，并转②。

④ 如果 M 中有转换，则对每个可发生的转换 t 执行以下各步。

(i) 计算 M 发生 t 转换以后的结果，记为 M'(若 $M(p)=\omega$，则 $M'(p)=\omega$)。

(ii) 如果从根结点到 M' 的路径上存在可覆盖的标识 M''，则对 $M''(p)$ 中每个满足 $M'(p) \geqslant M''(p)$ 的 p，用 ω 重置 $M'(p)$。

(iii) 将 M' 作为树的一个结点，从 M 到 M' 画弧，用 t 标记，并将 M' 标记为 new。

利用覆盖树可以分析 Petri 网 $PN=(N, M_0)$ 的一些性质。

① 当且仅当覆盖树中不出现含有 ω 的标识时，PN 有界，且 $R(M_0)$ 是有限的。

② 当且仅当覆盖树中每个标识的元素都为 0 或 1 时，PN 是安全的。

③ 当且仅当转换 $t \in T$ 不出现在覆盖树中时，t 为死的，即存在死锁。

④ 如果 $M \in R(M_0)$，则覆盖树中必定存在一个标注为 M' 的结点，满足 $M' \geqslant M$。

对于有界 Petri 网而言，覆盖树包含了其所有可达标识，因此，前面讨论的 Petri 网的所有性质的分析可以用覆盖树来解决。但是，由于 ω 的无限性，覆盖树不能解决可达性和活性问题。

【例 18-1】 对图 18-10 所示的 Petri 网构造其相应的覆盖树。

解：Petri 网的初始标识为 $M_0=(1,0,0)$，转换 t_2 和 t_4 都可能发生：发生 t_2 转换产生标识 $M_1=(0,0,1)$，在 M_1 下，不再产生新的状态转换，所以 M_1 标识为 dead；发生 t_4 转换产生标识(1,1,0)，该标识覆盖了标识 M_0，因此新标识为 $M_2=(1,\omega,0)$。在 M_2 下，转换 t_2 和 t_4 都可能发生：t_4 转换产生旧的标识 $M_5=(1,\omega,0)(M_5=M_2)$；$t_2$ 转换产生标识 $M_3=(0,\omega,1)$，在 M_3 下，只能发生 t_3 转换，产生旧的标识 $M_4=(0,\omega,1)(M_4=M_3)$。由此得到图 18-10 所示 Petri 网所构造出来的覆盖树，如图 18-11 所示。

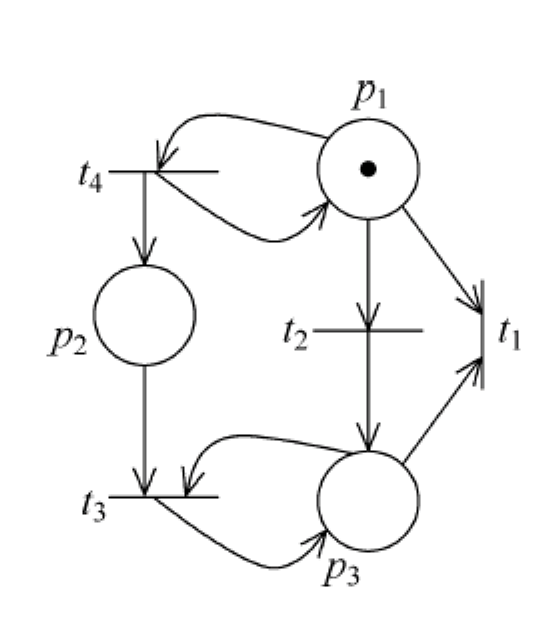


图 18-10 Petri 网

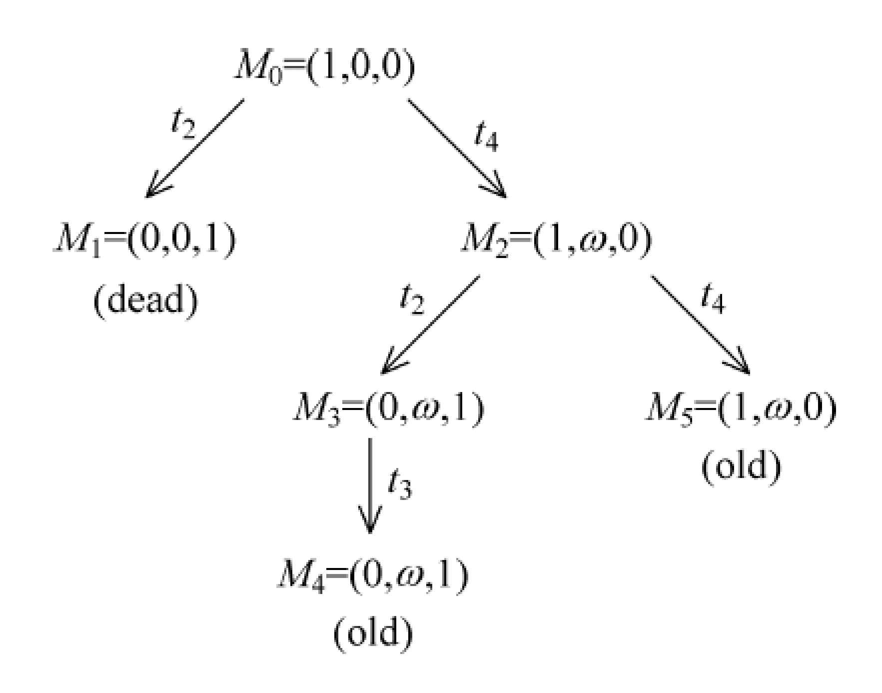


图 18-11 图 18-10 所对应的覆盖树

3. 不变量、关联矩阵和状态方程

(1) 不变量

Petri 网中如果有一些位置所包含的标记的总和在任何可达标识情况下均为常数，则这些位置就是系统的 S-不变量，以位置元素为序标的列向量表示。如果 Petri 网中有一些转换，它们的发生使标识恢复到它们的开始状态，则这些转换就是系统的一个 T-不变量，以转换元素为序标的列向量表示。

【例 18-2】 求如图 18-12 所示的 Petri 网的 S-不变量和 T-不变量。

解：图 18-12 所示的 Petri 网在任何标识的情况下，所包含的标记总数始终为 1，若以 I 表示系统的 S-不变量，则：

$$I = \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

转换 t_1、t_2 各发生一次，系统状态从 M 回到 M，则 t_1、t_2 是该系统的一个 T-不变量。同样 t_3、t_4 和 t_1、t_2、t_3、t_4 也都是系统的 T-不变量。

$$\begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{matrix}\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

其中，第一列为第一个 T-不变量，中间为第二个 T-不变量，两者线性组合为第三个 T-不变量。

【例 18-3】 图 18-13 所示的 Petri 网有没有 S-不变量？

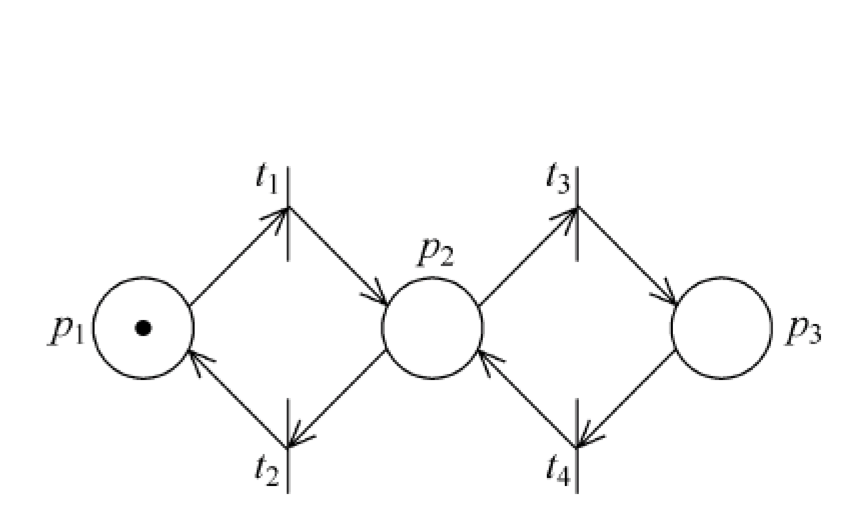


图 18-12 一个 Petri 网

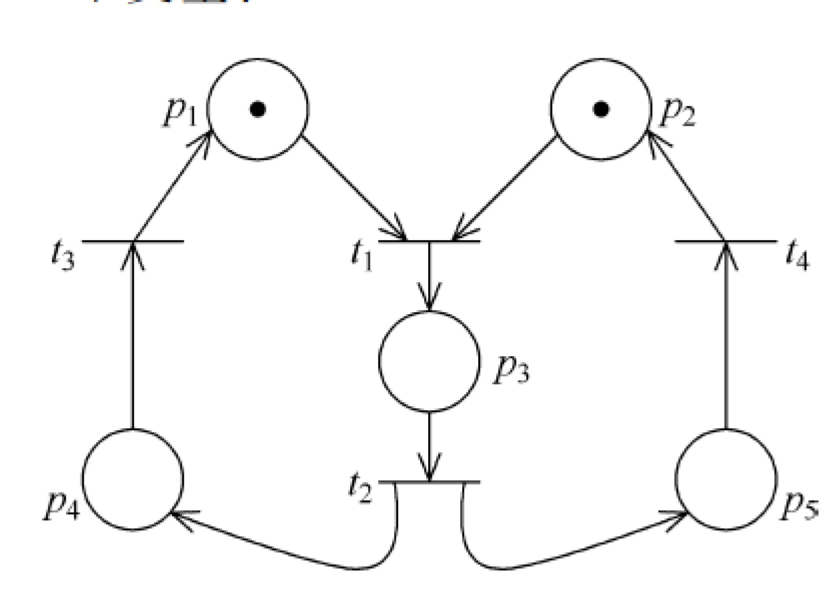


图 18-13 一个 Petri 网

解：初始状态下的标记总和为 2，初始状态下发生 t_1 转换时状态 p_3 获得标记，此时标记只有一个，因此该系统没有 S-不变量。如果状态 p_3 获得标记时认为是两个标记，则系统有 S-不变量：

$$I = \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{matrix}\begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \\ 1 \end{bmatrix}$$

【例 18-4】 图 18-14 所示的 Petri 网有没有 T-不变量？

解：转换 t_1、t_2、t_4 各发生一次，系统状态从 M_0 回到 M_0，则 t_1、t_2、t_4 是该系统的一个 T-不变量。同理，t_1、t_3、t_4 和 t_1、t_2、t_3、t_4 也是该系统的 T-不变量。如果 t_2 或 t_3 发生 2 次，其他转换各发生 1 次，也可以从 M_0 回到 M_0，这就有了另外两个 T-不变量，若以 J 表示系统的 T-不变量，则这两个 T-不变量分别表示为：

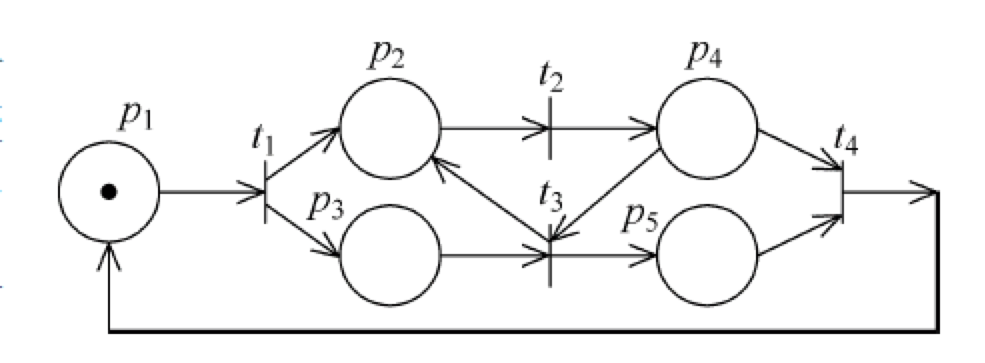


图 18-14 一个 Petri 网

$$J_1 = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{matrix}\begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \end{bmatrix}, \quad J_2 = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \end{bmatrix}$$

(2) 关联矩阵

Petri 网的结构可以用一个矩阵来表示。

设 Petri 网 $PN=(P,T,F,K,W,M_0)$，以 $P\times T$ 为序标的矩阵 C 表示该 Petri 网的关联矩阵，矩阵元素 $c(p_i,t_j)=W(t_j,p_i)-W(p_i,t_j)$，也可写成：

$$[c_{ij}]_{m\times n} = [c_{ij}^{+}]_{m\times n} - [c_{ij}^{-}]_{m\times n} \text{ 或者 } C = C^{+} - C^{-}$$

其中：

$c_{ij}^{+}=W(t_j,p_i)$ 是从转换 t_j 到它的输出位置 p_i 的弧的权；

$c_{ij}^{-}=W(p_i,t_j)$ 是从转换 t_j 的输入位置 p_i 到转换 t_j 的弧的权。

为了便于讨论，弧的权全部取 1，因此：

$$c_{ij}^{+} = \begin{cases} 1 & \text{当}(t_j,p_i)\in F \\ 0 & \text{否则} \end{cases}$$

$$c_{ij}^{-} = \begin{cases} 1 & \text{当}(p_i,t_j)\in F \\ 0 & \text{否则} \end{cases}$$

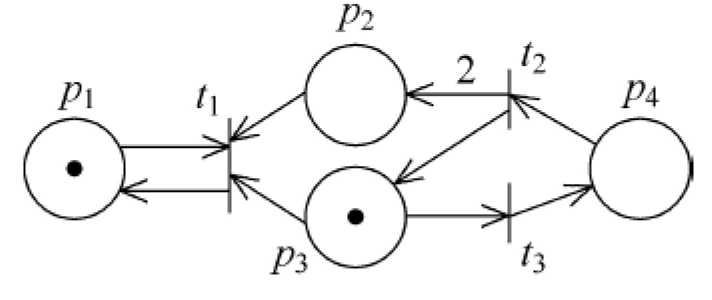


图 18-15 一个 Petri 网

显然，c_{ij}^{-}、c_{ij}^{+} 和 c_{ij} 分别表示当转换 t_j 发生，位置 p_i 中标记被取走、增加和改变的数量，关联矩阵表示取走、增加后产生的变化数，此关联矩阵给出了 Petri 网的结构。

如图 18-15 所示的 Petri 网对应的关联矩阵为：

$$\boldsymbol{C}=C^{+}-C^{-}=\begin{bmatrix}1&0&0\\0&2&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&0\\1&0&0\\1&0&1\\0&1&0\end{bmatrix}=\begin{bmatrix}0&0&0\\-1&2&0\\-1&1&-1\\0&-1&1\end{bmatrix}$$

(3) 状态方程

对于一个 Petri 网，如果 $M_1[t_j>M_2$，则标识向量 M_2 和 M_1 之间的关系可用式(18-2)描述：

$$M_2=M_1+C_{..j} \tag{18-2}$$

类似地，若 $M_0[\sigma>M$，则标识向量 M_2 和 M_1 之间的关系可用式(18-3)描述：

$$M=M_0+C\cdot U \tag{18-3}$$

其中：U 是 Petri 网的 T-向量，对于 $t_i\in T$，$U(t_i)$对应于 t_i 在 σ 中出现的次数(称为转换的发生向量)。称式(18-3)为 Petri 网状态方程。

(4) 状态方程在可达性分析方面的应用

状态方程为部分解决可达性分析问题提供了依据。对于 Petri 网 $PN=(N,M_0)$，若 M_d 从 M_0 可达，即 $M_d\in R(M_0)$。则 $M_d\geqslant 0$，且必存在转换发生序列 $\sigma=t_1t_2\cdots t_n$，这些转换依次发生后，标识从 M_0 变成 M_d，令 u_k 是第 k 个元素为 1，其余元素为 0 的发生向量，代入 $\Delta M=M_d-M_0=C\cdot X$，迭代 1 次得到：

$$X=\sum_{k=1}^{l}u_k\geqslant 0$$

向量 X 中各元素即为所对应转换的发生次数。因此，方程 $\Delta M=M_d-M_0=C\cdot X$ 必然存在一个非负整数解，该解即为转换发生的次数向量。

图 18-15 所示的 Petri 网中有：

$$\boldsymbol{C}=\begin{bmatrix}0&0&0\\-1&2&0\\-1&1&-1\\0&-1&1\end{bmatrix}\quad \boldsymbol{M}_0=\begin{bmatrix}1\\0\\1\\0\end{bmatrix}$$

在状态 M_0 下，转换 t_3 发生，得到标识向量 M_1：

$$\boldsymbol{M}_1=\begin{bmatrix}1\\0\\1\\0\end{bmatrix}+\begin{bmatrix}0&0&0\\-1&2&0\\-1&1&-1\\0&-1&1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}=\begin{bmatrix}1\\0\\0\\1\end{bmatrix}$$

发生序列 $\sigma=t_3t_2t_3t_2t_1$ 对应的发生次数向量为 $X=(1\ 2\ 2)^{\mathrm{T}}$，在 M_0 下发生 σ，得到标识向量 M 如下：

$$\boldsymbol{M}=\begin{bmatrix}1\\0\\1\\0\end{bmatrix}+\begin{bmatrix}0&0&0\\-1&2&0\\-1&1&-1\\0&-1&1\end{bmatrix}\begin{bmatrix}1\\2\\2\end{bmatrix}=\begin{bmatrix}1\\3\\0\\0\end{bmatrix}$$

对于从 M_0 可达的标识向量 $(1\ 8\ 0\ 1)^T$，方程

$$\begin{bmatrix}1\\0\\1\\0\end{bmatrix}+\begin{bmatrix}0&0&0\\-1&2&0\\-1&1&-1\\0&-1&1\end{bmatrix}X=\begin{bmatrix}1\\8\\0\\1\end{bmatrix}$$

存在一个解 $X=(0\ 4\ 5)^T$，它对应于发生序列 $\sigma=t_3t_2t_3t_2t_3t_2t_3t_2t_3$。

对于向量 $(1\ 7\ 0\ 1)^T$，方程

$$\begin{bmatrix}1\\0\\1\\0\end{bmatrix}+\begin{bmatrix}0&0&0\\-1&2&0\\-1&1&-1\\0&-1&1\end{bmatrix}X=\begin{bmatrix}1\\7\\0\\1\end{bmatrix}$$

无解，所以向量 $(1\ 7\ 0\ 1)^T$ 为不可达标识向量。

状态方程有解只是可达性的必要条件而不是充分条件，这是由于 ΔM 缺少初始标识信息导致的。

18.2.4 实例：ATM 系统

1. 问题的提出：用户需求

首先要弄明白用户需要什么样的系统，即系统功能和运行环境。

对 ATM 系统进行行为域的分析，得到如图 18-16 所示的状态图。

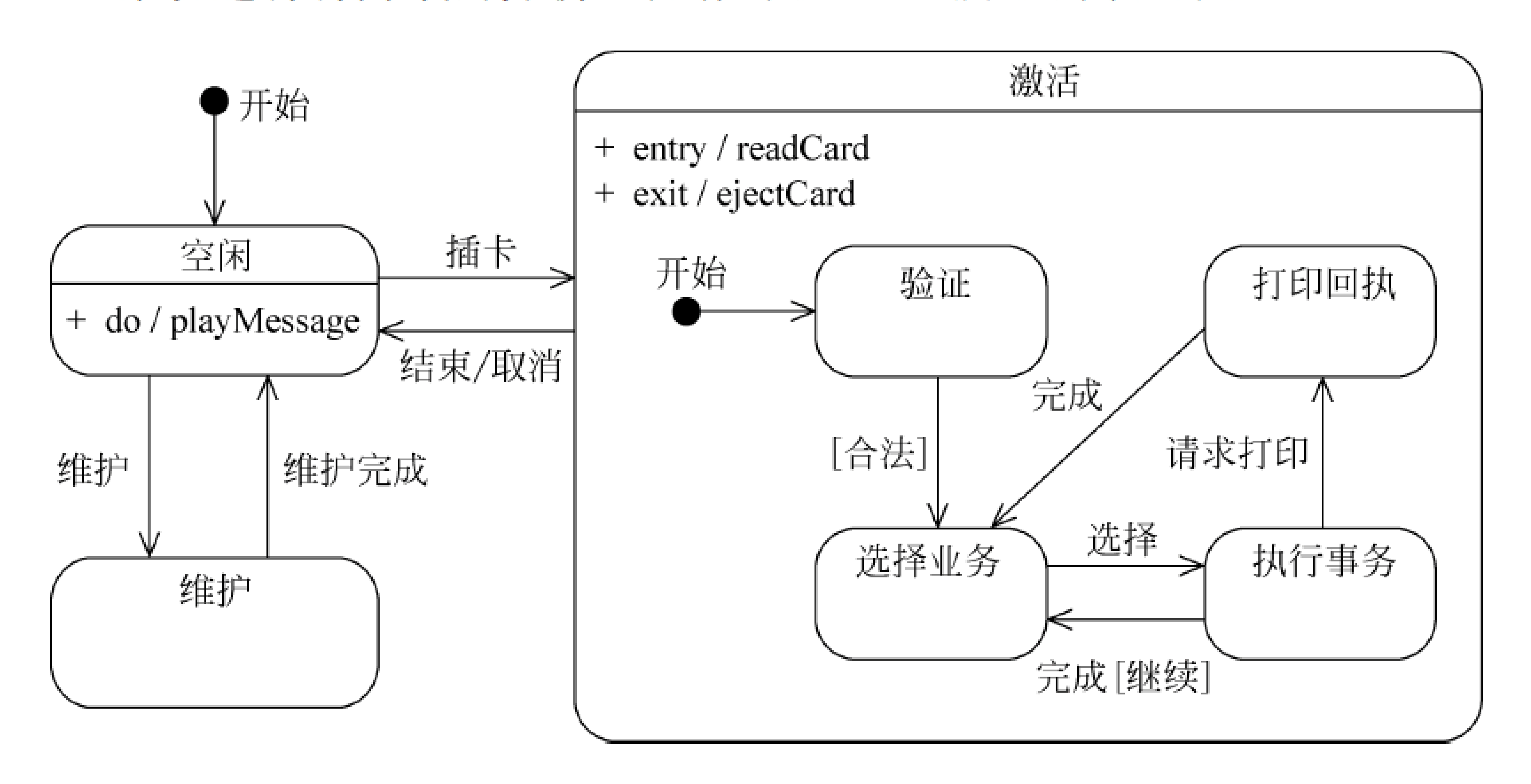


图 18-16 ATM 的状态图

2. 从需求到规格说明

用 Petri 网来表示 ATM 系统的状态图，如图 18-17 所示。

形式化定义：$PN=(P,T,F,M_0)$，其中：

p_1：ATM 柜员机空闲

p_2：ATM 柜员机验证卡是否有效

p_3：ATM 柜员机等待用户选择业务

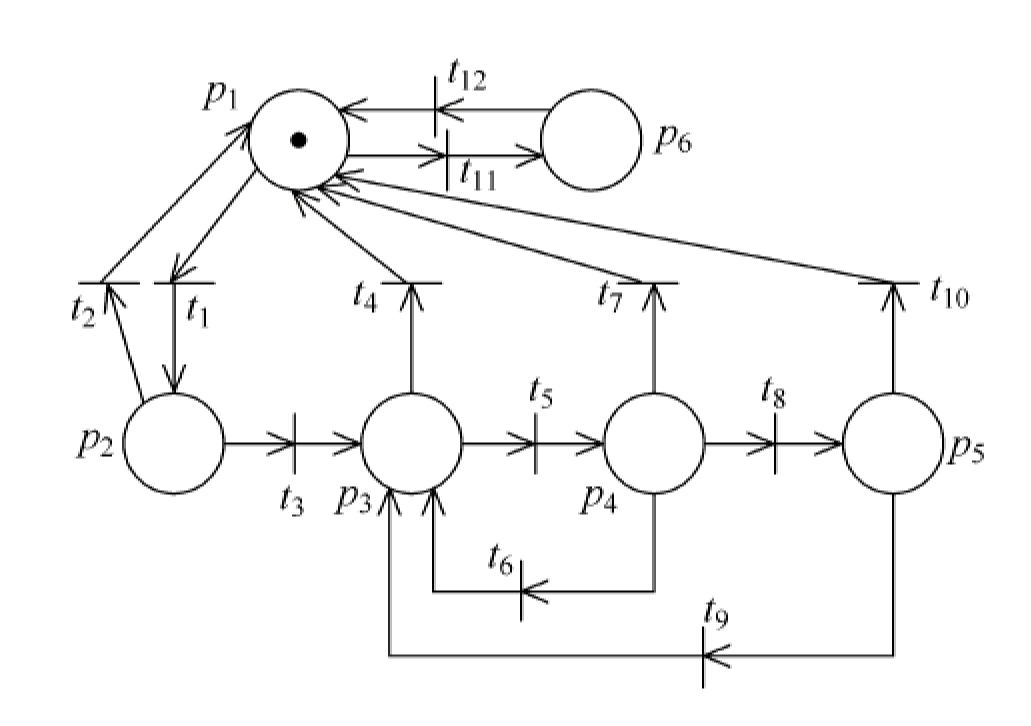


图 18-17　ATM 系统状态图对应的 Petri 网

p_4：ATM 柜员机执行具体事务

p_5：ATM 柜员机进行打印回执处理

p_6：ATM 柜员机正在维护

t_1：用户插卡

t_2：用户选择取消退卡或卡验证没通过而结束操作

t_3：卡验证通过

t_4：用户选择退卡而结束操作

t_5：用户选择具体业务

t_6：ATM 柜员机处理完事务后用户选择继续操作回到选择页面等待用户选择业务

t_7：ATM 柜员机处理完事务后用户选择退卡而结束操作

t_8：ATM 柜员机处理完事务后用户选择打印回执操作

t_9：ATM 柜员机完成打印回执操作后用户选择继续操作回到选择页面等待用户选择业务

t_{10}：ATM 柜员机完成打印回执操作后用户选择退卡而结束操作

t_{11}：工作人员选择维护

t_{12}：ATM 柜员机维护完成

3. 实现

系统中的变迁可以由赋值语句完成，状态则可以用布尔变量实现。

18.3　Z 语言

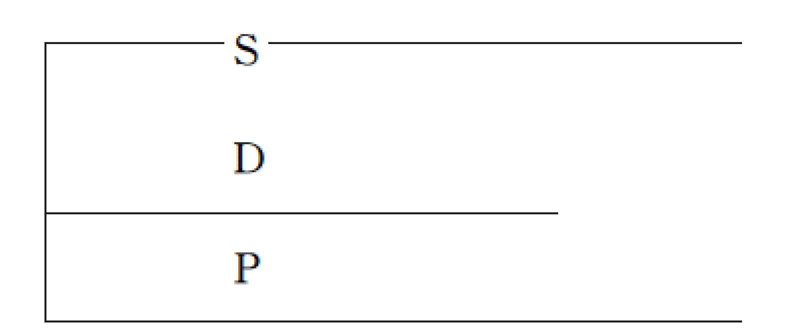
18.3.1　基本概念

Z 语言是在过去近 30 年里发展起来的规格说明语言，已在形式化方法领域中广泛使用。Z 语言在一阶谓词逻辑中应用集合、关系及函数来构造模式。表示抽象和操作抽象是 Z 方法中重要的两个方面。表示抽象，又称为数据抽象，是从数据结构的表示细节中抽象出

抽象数据结构(如关系、函数、集合、序列、包等),而不关心这些抽象数据结构在计算机中是如何表示和实现的;操作抽象,又称为过程抽象,指忽略任务具体完成的过程,只精确描述该任务所要完成的功能,即描述了从输入到输出的映射,该映射的定义域和值域均使用数据抽象来刻画。

模式是Z语言的基本结构,刻画了系统的静态性质和动态行为,可以描述状态、操作等。状态模式定义了目标软件系统某一部分的状态空间及其约束性,通过相应的抽象数据类型的变量以及在其上定义的约束关系进行描述;操作模式描述了系统某部分的行为特征,通过描述在操作之前该部分的状态值与操作之后该部分的状态值之间的关系来定义该操作的特征。一个模式有一个唯一的名字(S),并包括一个声明部分(D)和一个断言部分或谓词部分(P)。模式名可以在规格说明中随处调用,也可以作为一个类型名使用;声明部分引入变量及其类型,这些变量是该模式内的局部变量,声明部分形式为 $x:T$,其中 x 是变量,T是类型;断言部分描述了在这些局部变量之间,或者局部变量与在该模式之前声明的全局变量或常量之间的不变式关系。模式有垂直和水平两种形式[①]:

垂直形式:



水平形式:$S \triangleq [D \mid P]$,其中“$\triangleq$”为定义符号。由于垂直形式具有更好的可理解性和可读性,使用得更为普遍。

如下为一个简化的学生档案系统描述的模式 StudentSys:

[Student]



其中[Student]引入了一个基本类型“Student”。StudentSys为模式名,该模式的声明部分声明了两个变量:enrolled和tested,分别表示被录取的学生的集合和所录取的学生中经过测试的学生的集合;**P** 表示幂集。断言部分用谓词描述了两个关系:所录取的学生人数不超过size,经过测试的学生必须是被录取的学生;“||”表示集合的基数;断言部分的不同行谓词之间是合取关系(∧)。

下面定义ATM系统中的一个模式Account。

---

① 古天龙.软件开发的形式化方法.北京:高等教育出版社,2005

Account

账户：字符型
户名：字符型
身份证号码：字符型
开户日期：日期型
开户网点：字符型
密码：字符型
账户余额：数值型

账户为 19 位数字
身份证为 15 位或 18 位数字
密码为 6 位数字
账户余额>0

在定义模式之后，就可以将其作为一个类型在其他模式中引用。例如，可以在模式 Accounts 的定义中引用刚才定义的 Account。

Accounts

all：**P** Account

$\forall a_1, a_2 : Account \mid a_1 \neq a_2 \wedge a_1 \in all \wedge a_2 \in all \wedge a_1.$账户$\neq a_2.$账户

其中，**P** 是集合的幂集运算符，使用运算符“.”可以指出模式中的某个成员。Accounts 的断言部分指出了类型 Account 的账户必须唯一。

18.3.2 模式运算

1. 模式扩充

对于给定模式 S，可以对 S 的声明部分和谓词部分分别进行扩充，“S; D”表示对 S 的声明部分使用 D 的声明部分进行扩充，得到的新模式的声明部分和声明 D 合并，而谓词部分是 S 的谓词部分；“S|P”表示对 S 的谓词部分用谓词 P 加以扩充，得到的新模式的谓词部分是 S 的谓词部分和谓词 P 的合并，而声明部分是 S 的声明部分。

例如，给定模式 $S \triangleq [x: N \mid x > 5]$，令 $S_1 \triangleq S;\ y: Z$，则有

S_1

x: N
y: Z

$x > 5$

令 $S_2 \triangleq S_1 \mid y < 0$，则有

S_2

x: N
y: Z

---

x>5
y<0

2. 模式包含

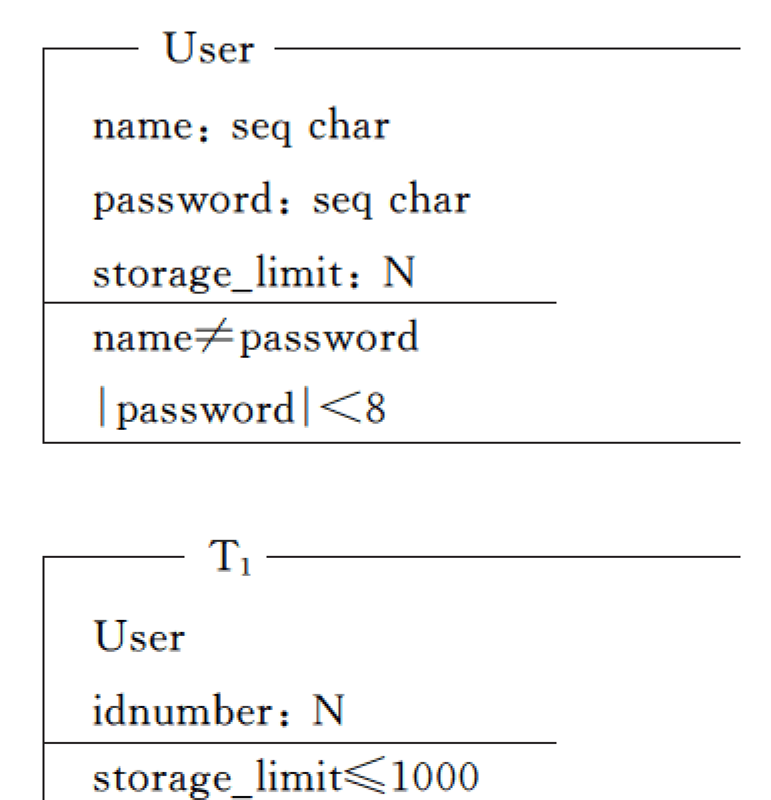
模式包含是指在一个模式的定义中通过模式名引用已经定义好的模式，从而使规格说明的长度大大缩短，并能够比较简明清楚地说明复杂的操作。设 S_1 和 S_2 是两个模式，若要在 S_2 的定义中引用模式 S_1，则要满足 S_1 和 S_2 是兼容的。这里的兼容是指，如果变量名 x 在 S_1 和 S_2 的声明中都出现，则 x 在两个模式中必须具有相同的类型。

模式包含可以通过以下两种方式实现。

① 模式 S_2 将模式 S_1 包含在其声明部分。

② 模式 S_2 将模式 S_1 包含在其断言部分。

例如，下面给出的模式 T_1，其声明部分包含了模式 User。



等价于下面的模式：

```
—— T1 ——————————
name: seq char
password: seq char
storage_limit: N
idnumber: N
————————————
Name≠password
|password|<8
storage_limit≤1000
————————————
```

3. 模式修饰

模式修饰是在模式名之后加上撇号“′”对该模式进行修饰，作用是将修饰应用到该模式中声明的所有变量上。模式修饰可以描述状态模式，刻画了状态转换前后的状态。修饰的状态模式在 Z 方法中成为后状态模式。即加上后缀“′”表示转换后的模式的状态及变量。

4. 模式连接

设 S 和 T 是两个模式，可以用布尔连接词“∧”、“∨”、“¬”、“→”、“→”组成新的模式：将

两个模式的声明部分合并，断言部分用给定的连接词连接构成新模式的断言部分。模式连接有一个前提条件，即 S 和 T 是兼容的。

5. 模式变量换名

模式变量换名是指对模式中的变量用新的变量名替换，记为：模式名[$y_1/x_1, y_2/x_2, \cdots, y_n/x_n$]，表示用 $y_1, y_2, \cdots, y_n$ 分别替换模式中的变量 $x_1, x_2, \cdots, x_n$ 的所有自由出现。

6. 模式变量消除

模式变量消除是使模式变量从模式的声明部分消除（即删除），同时在模式的断言部分将它们用存在量词进行量化，记为：模式名\($x_1, x_2, \cdots, x_n$)，表示对模式中的变量 $x_1, x_2, \cdots, x_n$ 进行消除，其中变量 $x_1, x_2, \cdots, x_n$ 必须出现在模式的声明部分。例如，下面给出模式 S 和对其进行变量消除之后的模式 T $\triangleq$ S\(x)。

```
┌──── S ─────────────────
│ x: N
│ S: PN
├────────────────
│ x∈S
└────────────────────────
```

```
┌──── T ─────────────────
│ S: PN
├────────────────
│ ∃x: N • x∈S
└────────────────────────
```

其中“•”用来分隔量词约束变量和谓词表达式。

7. 模式的Δ和Ξ表示

模式的 Δ 和Ξ表示是两类经常使用的模式的简写，可以使模式的描述及相应的规格说明更加简洁。对于模式 S，ΔS 是由状态模式 S 和相应的后状态模式 S′组合而成的。

```
┌──── ΔS ────────────────
│ S
│ S′
└────────────────────────
```

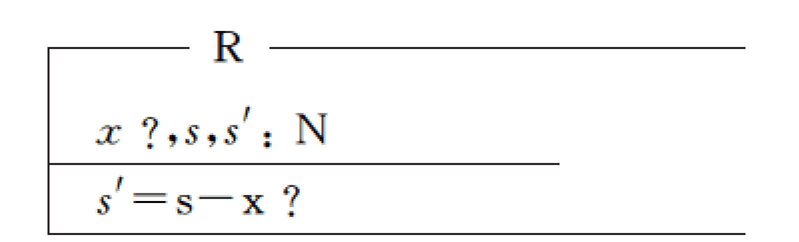
对于任意模式 S，ΞS 是操作模式，该操作不引起系统状态的任何改变。可以在 ΔS 的基础上将ΞS 定义为：

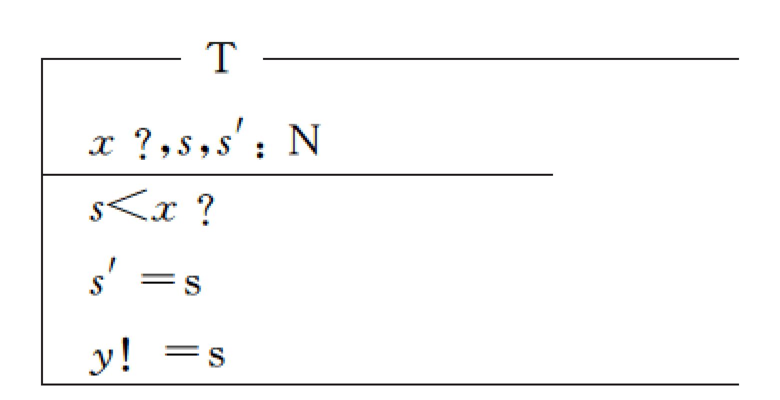
```
┌──── ΞS ────────────────
│ ΔS
├────────────────
│ No change
└────────────────────────
```

当对应于某个具体的模式时，No change 的位置是相应的一系列谓词，描述后续状态中的变量与前状态中的变量的相等关系。

8. 模式复合

模式复合是将一个模式的后状态变量与另一个模式的前状态变量进行关联，其中所关联的变量必须具有相同的基本名。为了区分输入变量和输出变量，在输入变量后加上后缀"?"，在输出变量后加上后缀"!"，不带这些后缀的变量名则成为变量的基本名。对于模式 B 和 C，如果模式 B 引起变量 s_1 改变为 s_2，模式 C 引起变量 s_2 改变为 s_3，则模式 B 和模式 C 的复合就会产生变量 s_1 到 s_3 的改变，并得到一个新的模式 A，记为 $A \triangleq B \circ C$。可以将模式 A 所描述的状态改变看成两个步骤的过程：引入一个中间变量作为 B 产生的结果，该变量又作为 C 的起始变量。以下面的模式 R 和 T 为例来看模式复合。





模式 R 和模式 T 复合的步骤如下。

(1) 检查两个模式中的"′"修饰变量及其非修饰变量的集合是否相同，如果不同，则模式复合是无定义的。本例中该集合为$\{s', s\}$，在 R 和 T 中相同。

(2) 检查两个模式的声明部分的输入和输出变量的类型是否一致，如果冲突，则模式的符合是无定义的。本例中模式 R 和 T 中都出现的输入变量为 $x?$，类型都为 N。

(3) 将模式复合中的前一模式内的后状态变量换名为一个新的变量，在本例中将 R 中的 s'换名为 s^+，即设 $R_1 \triangleq R[s^+/s']$。

(4) 将模式复合中的后一模式中的前状态变量换名为步骤(3)给出的新变量，在本例中将 T 中的 s 换名为 s^+，即设 $T_1 \triangleq T[s^+/s]$。

(5) 对步骤(3)、(4)中换名后所得到的模式进行合取，对于本例，有 $\alpha \triangleq R[s^+/s'] \wedge T[s^+/s]$。

(6) 对步骤(5)所得到的新模式，隐藏其中在步骤(3)和(4)中引入的新变量，对于本例，有 $\beta \triangleq (R[s^+/s'] \wedge T[s^+/s]) \setminus (s^+)$。

在各个步骤中所得到的模式如下：

R_1

$x?, s, s^+ : N$

$s^+ = s - x?$

T_1

$x?, s^+, s', y! : N$

$s^+ < x?$

$s' = s^+$

$y! = s^+$

α

$x?, s, s^+, s', y! : N$

$s^+ = s - x?$

$s^+ < x?$

$s' = s^+$

$y! = s^+$

β

$x?, s, s', y! : N$

$\exists s^+ : N \bullet$

$(s^+ = s - x?$

$s^+ < x?$

$s' = s^+$

$y! = s^+)$

对模式 β 进行简化后，最终得到模式：

β

$x?, s, s', y! : \mathbb{N}$

$s' = s - x?$

$s' < x?$

$y! = s'$

18.3.3 操作模式

在 Z 方法中，规格说明使用操作模式来对系统状态空间的变化进行描述，从而定义系统所完成的功能。操作模式的声明部分包括前状态变量、输入变量、后状态变量以及输出变量；断言部分通过前状态变量和输入变量的谓词来描述该操作所需满足的前置条件，通过后状态变量与前状态变量之间的关系来描述该操作的效果。

下面给出录取学生操作的模式 EnrollStudent。该操作输入学生的姓名，输出提示信息。执行该操作的前置条件是：所输入的学生是未录取过的，并且当前已经录取的学生人数少于 size。执行该操作产生的效果是：新的 enrolled 是原来的 enrolled 与新录取的学生的并集，而 tested 未发生变化，同时输出提示信息“success”。

Enrollstudent

ΔStudentSys

name?: Student

r!: Response

name? $\notin$ enrolled

$|$enrolled$| <$ size

enrolled$'$ = enrolled $\cup$ {name?}

tested$'$ = tested

r! = “success”

为了描述完整的操作，还需要给出在不满足前置条件时应当做出的相应处理。因此，对于上面的操作模式，还需定义另外两个操作：

ExistAlready

ΞStudentSys

name?: Student

r!: Response

name? $\in$ enrolled

r! = “same name exist”

SizeLimited

ΞStudentSys

r!: Response

$|$enrolled$| \geq$ size

r! = “too many students”

其中 Response：：=“success”|“same name exist”|“too many students”。

这样，一个完整的录取学生的操作可定义为：

DoEnrollStudent $\triangleq$ EnrollStudent $\vee$ ExistAlready $\vee$ SizeLimited

18.3.4 实例：ATM 系统

简化的 ATM 系统要完成以下事务处理。

(1) 银行卡的识别、验证。

(2) 选择操作并处理。

① 查询。

② 取款。

③ 转账。

④ 修改密码。

(3) 打印。

(4) 退卡。

下面对取款操作进行分析。

取款需要满足下述的约束。

① 取款金额必须少于 ATM 机器中的现金余额。

② 取款金额必须少于账户余额。

③ 每次的取款金额必须少于 1000 元。

对于取款操作，抽象出一个账户类型，用之前已经定义的模式 Account 来表示。当操作成功时，系统返回成功的信息；操作不成功时，系统返回相应的提示。用类型 Report 定义可能的提示信息。

Report：：=“成功”|“重新输入金额”

```
——Withdraw——————————————
△Account
取款金额?：数值型
ATM 余额：数值型
r!：Report
————————————————————————
账户余额'=账户余额-取款金额
取款金额<账户余额
取款金额<1000
取款金额<ATM 余额
r! =“成功”
————————————————————————
```

取款操作在以下 3 种情况下会出错：输入的取款金额大于 1000；输入的取款金额大于 ATM 机器中的现金余额；输入的取款金额大于账户余额。这 3 种情况下的操作分别用 Over Limit、CashShortage 和 BalanceShortage 这 3 种模式描述。

```
——OverLimit—————————————
ΞAccount
取款金额?：数值型
r!：Report
————————————————————————
取款金额>1000
r! =“重新输入金额”
————————————————————————
```

CashShortage

ΞAccount
取款金额?：数值型
ATM 余额：数值型
r!：Report

取款金额>ATM 余额
r! =“重新输入金额”

BalanceShortage

ΞAccount
取款金额?：数值型
r!：Report

取款金额>账户余额
r! =“重新输入金额”

取款操作的完整规格说明为

$$DoWithdraw \triangleq Withdraw \vee OverLimit \vee CashShortage \vee BalanceShortage$$

其他的操作与取款操作相类似，就不再一一列举。

18.4 形式化方法的优缺点

形式化方法的优点如下。

(1) 对系统的需求规格说明描述精确，定义完整。

(2) 形式化的需求规格说明有利于系统的设计与实现。

(3) 软件实现的正确性可以形式化验证，确保软件质量。

PRG 与 IBM 的 Hursley 实验室合作，将 Z 语言用于 IBM 的客户信息控制系统的开发，使最终产品的质量得到了全面提高，检测出的错误数量大大减少，并且整体开发费用降低了 9%。

但是软件形式化方法的理论和技术还未令人满意，在实用化、工程化方面仍有许多问题有待研究解决。形式化方法存在的主要缺点如下。

(1) 形式化的需求规格说明可读性差。

(2) 形式化方法对软件设计人员提出较高要求，需要进行更专业化的培训。

(3) 形式化方法只适用于能够静态定义的软件系统，它无法定义动态系统行为。

(4) 形式化的规格说明(形式语义模型)的正确性验证还不能简化或自动化。

(5) 形式化方法目前还缺乏软件工程环境的支持。

(6) 形式化规格说明主要关注于功能和数据，而问题的时序、控制和行为等方面难于表示。

(7) 有些问题元素,如人机界面,最好用图形技术或原型来表示。

近年来,形式化方法在以下两个方面的发展大大改善了其某些缺点。

(1) 形式化方法与图形语言机制相结合,兼具了图形表示的直观、简洁以及形式化方法的严谨、精确。

(2) 用 CASE 工具支持形式化软件开发,可以简化需求分析和需求描述工作,而且还可以利用自动定理证明技术帮助分析人员验证规格说明的数学性质。

实践证明,这两个技术对于克服形式化方法的缺陷是有效的,因此,它们将在形式化方法的未来发展中发挥重要作用。

18.5　形式化方法的发展

将形式化方法应用于软件开发过程,首先,在需求分析阶段的信息收集和信息分析两项工作中采用形式化的规格说明语言(如 Z、VDM、Larch 等)来构造严格的形式化需求规格说明,即形式语义。不同的形式化规格说明语言在表达能力、术语、准确性以及支持形式化处理的能力方面各有不同,要根据软件系统的功能特征和行为特征选择。然后,以该形式化需求规格说明为起点,借助相应的形式化开发支持工具辅助实现目标软件系统。目前,除了在软件设计、编码阶段采用形式化方法外,还开展了形式化测试的研究工作,软件形式化开发方法还有许多问题有待研究发展。形式化方法和面向对象方法的结合也是研究的一个热点:如何利用面向对象结构来提高形式符号的表达能力,如何使用形式化方法来分析面向对象的语义或提高这些标记符号表达对象概念的能力。另外,形式化方法和其他传统软件开发方法相结合以达到取长补短的目的,也是值得研究的课题。

本 章 小 结

形式化方法能够简洁而准确地描述需求规格说明、进行验证,还可以进行程序求精。它使所生成的模型比起传统的或面向对象的方法生成的模型更完整、一致、无歧义。

本章着重讨论了两种形式化方法——Petri 网和 Z 语言。Petri 网的理论和形式化分析方法可以帮助我们完成规格说明及其求精、综合和分析,本章仅介绍了简单的 Petri 网应用,对于一些复杂系统而言,会使用大量的位置和转换,由此引起状态组合爆炸问题,解决该问题的方法之一是使用谓词/转换网和着色网,有兴趣做进一步研究的读者可以查阅相关资料。

Z 方法是以一种基于集合理论和一阶谓词逻辑的形式化规格说明方法。表示抽象和操作抽象是 Z 方法规格说明的两个重要方面,模式是 Z 方法的基本结构,利用模式建立模块化的 Z 方法规格说明,并进行复合得到整个软件系统的规格说明。

思考与练习

1. 针对本章第 18.1.1 节中提到的几个非形式化方法的不足之处举出实例。

2. 在形式化开发方法中,从形式化规格说明到目标软件系统的可实现和可执行的角度来看,形式化方法可以分为哪几类?每一类的基本原理和可采用的技术有哪些?参考相关资料,了解除 Petri 网和 Z 语言以外的其他形式化语言。

3. 形式化开发方法一般有哪几个步骤?各个步骤的目的是什么?

4. Petri 网主要适用于哪类系统的建模和分析,它有哪些优点和不足?

5. 运用 Petri 网进行系统建模和分析的步骤是什么?

6. Petri 网可以分析系统的哪些结构特性和行为特性?这些特性的具体含义是什么?

7. 利用覆盖树可以分析 Petri 网的哪些性质,如何分析?

8. 自动售货机和电梯是日常生活中常见的自动控制装置,试分别建立 Petri 网的规格说明。

9. 对于某互联网应用系统,每个用户使用系统前必须登录。应用 Z 语言规格登录系统 LogSys,该系统可以保存已注册的用户及其口令信息,并且可以跟踪当前已登录系统的在线用户。

(1) 定义 ΔLogSys、ΞLogSys 和 InitLogSys。

(2) 定义操作模式,用来注册一个新的用户和口令。

(3) 定义操作模式,用来注销一个用户及其口令。

(4) 定义操作模式,用来让一个用户登录。

(5) 定义操作模式,用来让一个已登录的用户修改口令。

(6) 定义操作模式,用来让一个已登录的用户退出系统。

某邮局能处理 3 种类型的邮件:平信、挂号和快件。邮件可能在不同时间从不同的地方到达邮局。邮局经过 3 个步骤分类:第一个过程是将邮件按将要发送的地区分类(可以用邮编的前 3 位数标识);第二个过程是按地区内的位置分类(可以用邮编的后 3 位数标识);第 3 个过程是按街道号分类。经过第三个过程的分类后,再把邮件按其邮寄类型划分。最后,邮件就可以由邮局寄出。借助 Z 语言对邮局的邮件处理系统进行规格化说明。

10. 一个小型的图书馆系统需要完成以下事务处理。

(1) 图书的借出、归还。

(2) 向书库添加图书或从书库中删除图书。

(3) 根据图书的作者或图书的主题查询图书信息。

(4) 查询某个读者的借书信息。

(5) 查询某本图书的最后借阅人。

该图书馆系统有两类用户：管理员和读者。管理员可以进行上述的所有操作；读者可进行事务(3)的操作，也可以通过事务(4)查询自己的借书信息。

该系统还需要满足下述约束。

(1) 图书馆中所有的图书都能被借阅。

(2) 已借出的书不能再被借阅。

(3) 读者所能借阅的图书总数不能超过给定的限制。

请应用 Z 语言给出该系统完整的操作模式。



附录A 软件工程和知识可视化表征

按我国学术界的一般看法，知识是对事物属性与联系的认识，表现为对事物的知觉、表象、概念、法则等心理形式[①]。就它的反映内容而言，是客观事物的属性与联系的反映，是客观事物在人脑中的主观映像。就它的反映活动形式而言，有时表现为主体对事物的感性知觉或表象，属于感性知识，有时表现为关于事物的概念或规律，属于理性知识[②]。

一、理解知识可视化

在漫长的文明史中，语言就是用来描述知识和概念关系的。很多句子都能很轻易地用文字描述概念和它们之间的关系。人们多数时候用书写和言语来交流。但我们生活在一个视觉化的世界中，视觉是人类感觉中最精确的。用图画或图解来描述信息，可以达到理解上的突破，它是将知识复杂的语义和相互关系降减、转译成可视化的图形。知识工程和知识表征研究近年来开始关注用知识启发方法和外形符号来表征知识。例如，破解 DNA 结构的秘密是得益于可视化地呈现了"双螺旋"模型。

"可视化"一词来源于英文"Visualization"，原意是"可看得见的、清楚的呈现"，也可译为"图示化"。可视化作为一个专门研究领域是从多个与计算机相关的学科中发展起来的，其基本思想是用图形、图像等视觉化方式来表示大量数据，是数据或概念的图形化描述。一般来说，知识可视化是研究如何运用视觉表征促进知识在人们之间的创新和传播的理论、技术和方法，是所有用来建构和传达复杂意识的图形形式[③]。把可视化手段引入知识表征过程，无疑具有极其重要的意义。

根据美国心理学家 Paivio 提出的双重编码理论，在信息的储存、加工与提取中，语言与非语言的信息加工过程是同样重要的。这一理论的最重要的原则就是可通过同时用视觉和语言的形式呈现信息来增强信息的回忆与识别。基于双重编码理论的假设，可以做出这样的假设：如果将知识以图解的方式表示出来，那么将为基于语言的理解提供有效的辅助和补充，便于整体性地理解知识。

从知识的学习过程来看，"隐性知识"要能够转化为"显性知识"才能够被记录保存，这个过程叫做隐形知识的"表达外化"。而"显性知识"则经过人类大脑的综合组织，被作为"隐形知识"而保存在脑中(附图 A.1)。知识形态之间的转化，需要一种视觉化模型来表达和呈

---

① 顾明远. 教育大词典. 上海：上海教育出版社，1991：144

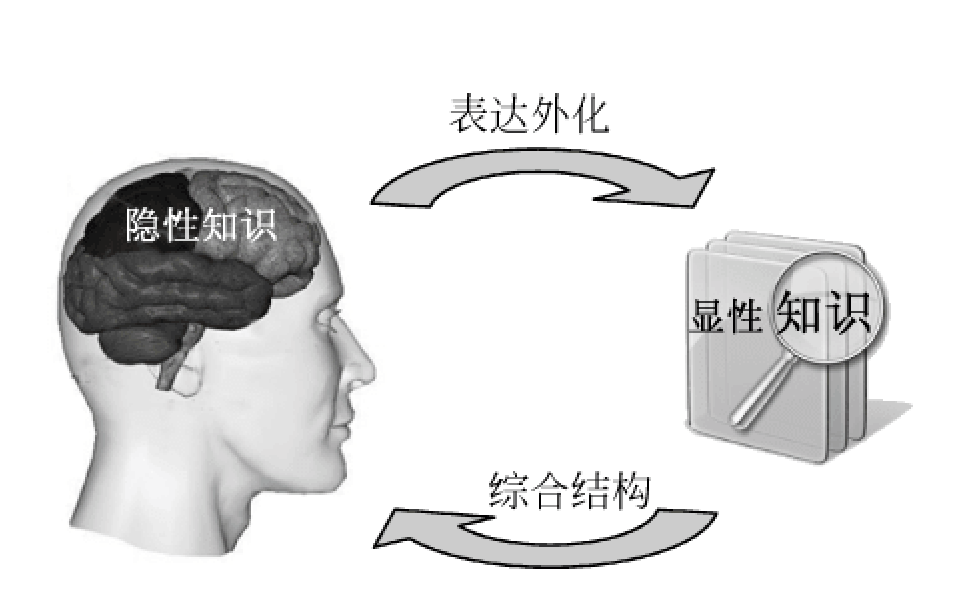
② 董纯才. 中国大百科全书. 北京：中国大百科全书出版社，1985：525

③ Eppler M. J.，Burkard R. A.. Knowledge Visualization：Towards a New Discipline and its Fields of Application. ICAWorking Paper，University of Lugano，2004

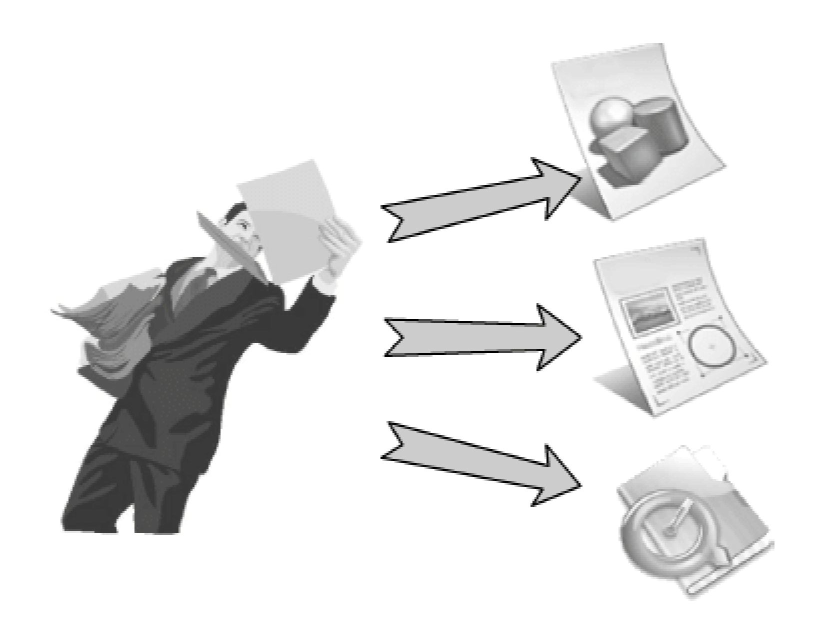
现。实际上，在软件工程领域的软件需求与分析、设计阶段，早已采用 UML 作为可视化建模工具，用于与客户和团队交流。

就学习来说，知识可视化是有效的支持工具之一。教学材料中的信息如果同时通过言语和视觉两条通道输入大脑，记忆会变得容易而持久。它有利于学习者极大地提高获取和运用知识的水平，将模糊不清的思想转变成清晰的外在形态，制约不必要的认知工作并创建新的知识结构。有些知识可视化形式可以通过降减信息的复杂度和相关的认知负荷，以支持人类复杂的认知过程。

由以上介绍可知，知识可视化的对象是人类知识，其表现形式要比数据可视化和信息可视化要丰富，因为它要表现的是人的智力加工后的知识制品，所以与数据可视化和信息可视化的计算机图形、图像不同，它必须是由人的智力直接参与绘制的图形图像，要体现的是人对于所表现的知识的理解、意见等。知识可视化形式可以是人工绘制的图形，也可以借助计算机软件绘制。知识可视化的目的是为了促进群体间知识的传播和创新，从这个角度看，知识可视化的交互类型是人-人交互，其核心是将知识转换成视觉形式(附图 A.2)。



附图 A.1　隐性知识和显性知识的转换

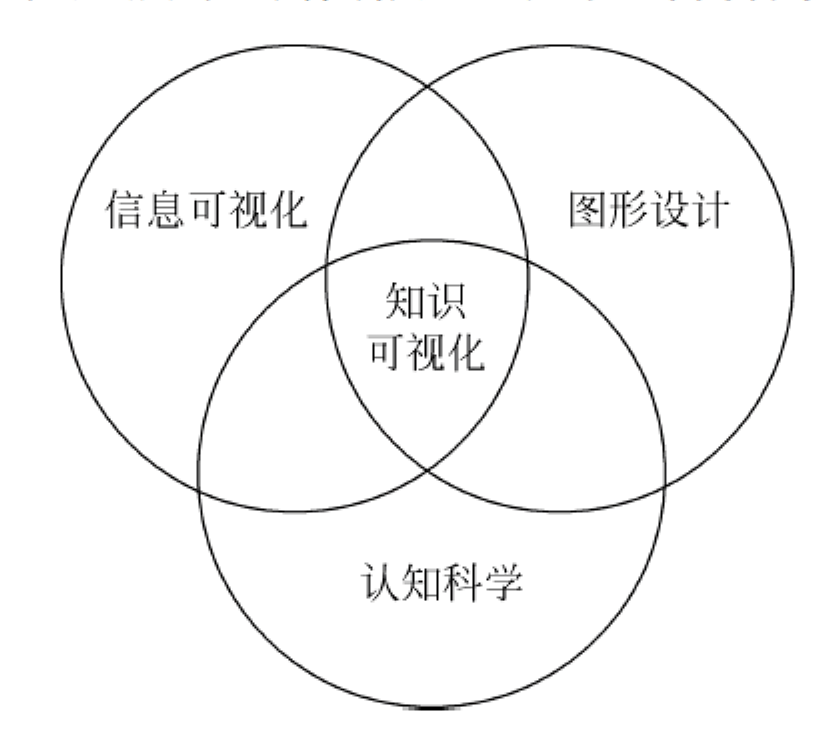


附图 A.2　将隐性知识转换成显性知识的可视化

二、知识可视化与知识表征

知识可视化是一种“知识表征”，是指知识的外在表现形式，它是承载知识的图解手段，也是直接作用于人的感官的刺激材料。

知识表征(Knowledge Representation)在认知心理学、人工智能以及知识可视化等领域有着不同含义。在认知心理学中，知识表征是指知识在人脑中的存储和组织形式或者说是知识在人脑中的呈现方式，与此相对应的是人脑的各种记忆模型；在人工智能领域，知识表征更多地被称为知识表示，是指知识在计算机中的存储形式和运算机制，因此与其对应的是数据结构和相应的算法；而在知识可视化领域，知识表征则是指知识的外在表现形式，与此相对应的是承载知识的图解手段，也是直接作用于人的感官的刺激材料。由此可见，知识可视化是连接认知心理学和人工智能研究的新的桥梁，为计算机的知识表征作用于人脑提供了可依赖的方法和手段，如附图 A.3 所示。



附图 A.3　知识表征在相关领域的含义[①]

① http://elitesol.springnote.com/pages/1586466

三、软件工程与可视化

在计算机诞生的初期，可视化就随着软件出现应运而生了。大概还是汇编语言刚问世不久，编程人员就开始绘制流程图来做辅助工具，使程序设计更加方便快捷。之后，随着编程语言的升级(出现高级语言)和处理对象规模扩大，出现了各种图形辅助工具，例如，图形用户界面技术、可视化操作等。可以说，可视化是伴随着计算机科学和技术的发展与应用而发展起来的。

一般说来，软件系统，特别是大型软件系统，都是极为复杂的系统，而软件系统最终的表现形式必为可执行代码。而最终可执行代码则更为复杂，包含了更多的细节，以至于再高明的软件人员也难于把握系统的全貌。这时就需要在软件总体构架与最终目标代码之间有一个中间过程，从而保证软件系统具有一致性和可理解性。模型是对现实世界的复杂系统的简化和抽象，而可视化模型又可以把复杂模型简单化和直观化。因此，可视化模型就是这个中间过程。由于可视化的分析建模能够实现将现实世界直接映射到软件模型之上，可以适应业务需求的不断变化，因此，可视化建模可以帮助软件设计人员加深对系统的认知，做到抓住问题的本质，并且滤掉众多非本质因素，从而有利于问题的解决。

在软件系统开发过程中，保持系统设计与代码的一致性是软件开发的一大难题，在软件的开发过程中，经常出现编码与系统设计不一致，而没有及时修改设计的现象，如果这种情况不断发展，就会导致系统设计与实际代码功能逐渐脱节，给以后的系统维护埋下隐患。但是，如果靠手工维护，又会异常费时费力。而在可视化建模技术中，由于建模语言没有语义的歧义性，可以由软件工具实现模型与代码之间的同步，从而保证模型和代码的一致性。

从技术实现的角度看可视化建模，可以发现它有如下很多好处。

(1) 可以有效管理系统的复杂度。一个现实世界的实体系统，往往是非常复杂的，相应地，它的对象模型经过简化，但其复杂度仍然非常大，有时甚至超出手工的处理能力。面向对象方法的最大优点就是抽象，通过可视化建模的抽象，使概念系统大为简化，直至达到人们能够理解和处理的程度。特别地，可视化系统是一个层次系统，开发人员可以根据需要了解任何一个层次，粗到系统架构，细到最深层次的细节。

(2) 可以实现开发人员之间很好的沟通。语言和文字是人们进行交流的主要手段，但是，语言和文字往往有歧义性，较难保证交流双方的理解完全一致。所以在工程技术领域，人们更多的是使用各种各样的模型进行交流和沟通。因此，可视化模型能够比较好地保证交流者之间对问题理解的一致性。

(3) 可以提高系统设计的可重用性。可视化建模能够实现每个类的功能单一化和构件化，因而，很多构件都有重用的机会，从而提高系统设计和开发效率，降低成本。

(4) 增强系统的灵活性。应用可视化技术建立的模型结构清晰、易于被理解，因而，系统比较易于修改和重构。

随着可视化建模技术不断广泛应用，出现了很多支持可视化建模的工具，例如，在软件工程领域使用的 IBM Rational 的 Rose 和 Microsoft Visio 等。这些工具可以全面支持 UML 视图建模。除此之外，还有一些用于认知科学与教育心理学的知识可视化的表示工具也可以用在软件工程分析与建模上，为软件建模与知识表征提供了新的方法与工具。

四、知识可视化的表示工具

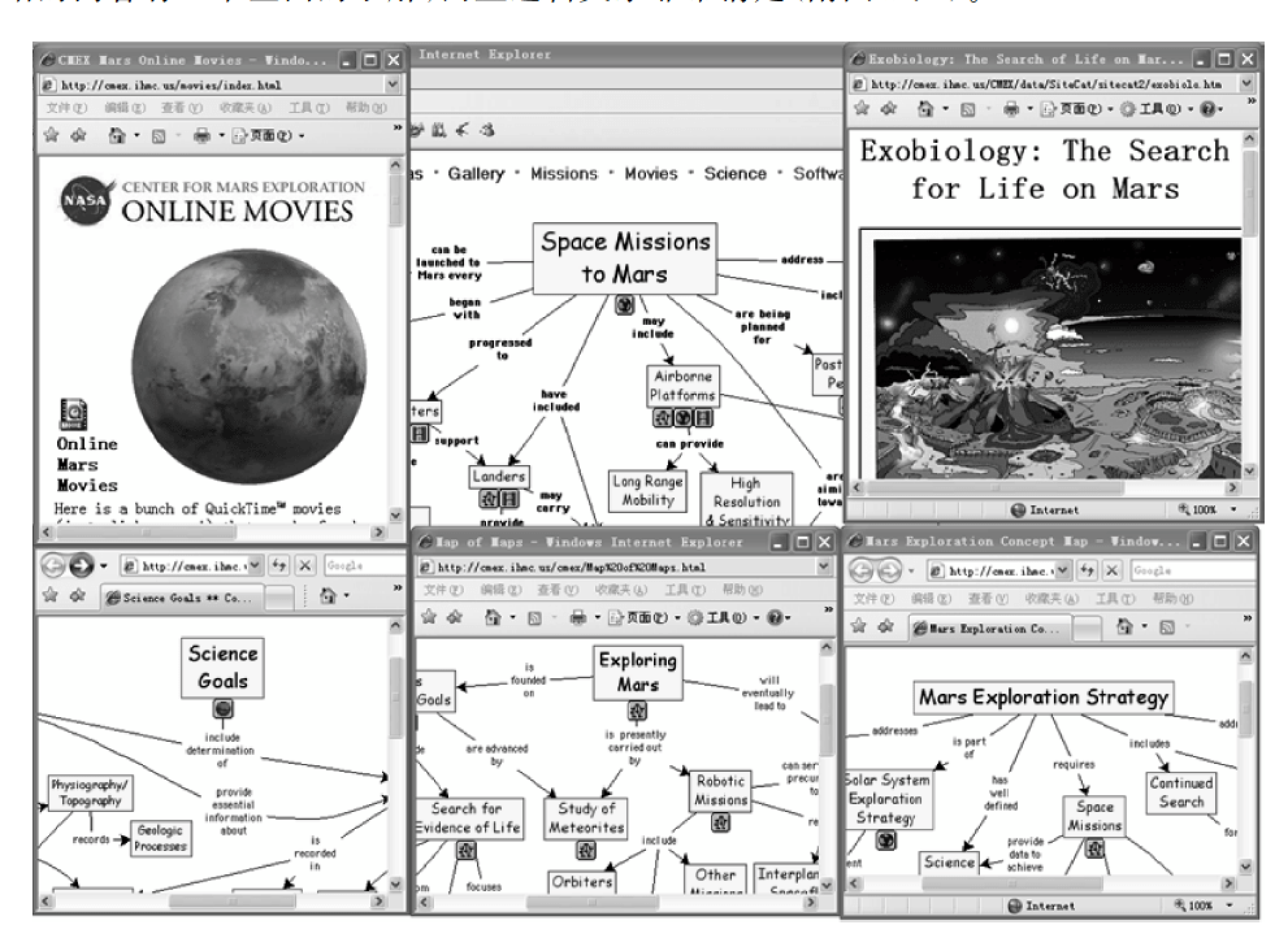
知识可视化包括所有能够建构和传递复杂思维的图解形式，包括图表、矩阵、概念图、知识图等，这些图解形式各有内容和形式上的特点。在内容方面，它们表达的不仅是信息与数据，更多的是思维、原理及事物间的关联。

目前在有关学科中，采用的知识可视化的表示工具主要有以下几种。

（一）概念图

概念图(Concept Map)是用来组织和表征知识的工具。它通常将某一主题的有关概念置于圆圈或方框之中，然后用连线将相关的概念和命题连接，连线上标明两个概念之间的意义关系。这种知识可视化方法最大的优点在于把知识的体系结构(概念及概念之间的关系)一目了然地表达出来，还突出表现了知识体系的层次结构。另外，概念图还是很好的结构化知识评估工具。

Web技术的发展为知识可视化和信息可视化进行整合创造了条件。美国航空航天局(NASA)的火星探测中心(CME)已经将其火星探测计划以“概念图”的方式发表在网络上。在此“概念图”中，主题“火星探测”(Exploring Mars)出现在一个首页面的中心，按照概念图的结构不同的内容依据分类互相连接，每个概念结点之下附有链接关系的图标。点击任何一个链接，首先会出现一段简短的浮动文字说明，让读者决定是否有兴趣点击进入并进行进一步的阅读。点击进入后会有另一个网页窗口打开，使读者不用离开概念地图即可对所要介绍的内容有一个全面的了解，而且逻辑关系非常清楚(附图A.4)。



附图A.4　美国NASA将概念图应用于火星探测的规划上

对于火星探测这样一个宏大的项目或主题，用一个简单的概念图很难表达复杂的概念知识并让用户去理解。基于 Web 形式发布的 NASA 的“火星探测”概念图已经扩展了传统概念图的使用，超过了只限于知识表示的用法。为了使这个庞大主题的知识描述更便于表示，NASA 把这个大的主题的表示法分成多个集合，每个集合形成一个概念图。为了显示这些概念图之间的关系，设计者通过导航、链接的方式把它们链接起来，可以从一个概念图链接到另一个概念图。另外，还可以链接到其他的资源，如图片、视频、声音、文本、Web 页面等，帮助解释和理解在地图中的信息，这样就形成了一个更大的概念图，从而形成整个关于“Exploring Mars”主题的概念图。通过 NASA 的“Exploring Mars”概念图和信息可视化的有效整合，可以看出这种可视化的表示方法功能更强大、更有效。当然，这只是知识可视化和信息可视化整合的一个例子，通过不断地探索研究，知识可视化和信息可视化的整合将会带来更好的前景。

（二）思维导图

思维导图（Mind Map），也称为心智图，最初是 20 世纪 60 年代英国人 Tony Buzan 创造的一种笔记方法。思维导图是一种将放射性思考（Radiant Thinking）具体化的方法。它依据全脑的概念，按照大脑自身的规律进行思考，全面调动左脑的逻辑、顺序、条例、文字、数字以及右脑的图像、想象、颜色、空间、整体思维，使大脑潜能得到最充分的开发，从而极大地发掘人的记忆、创造、身体、语言、精神、社交等各方面的潜能。美国波音公司在设计波音 747 飞机时就使用了思维导图。波音公司的 Mike Stanley 宣称，如果使用普通的方法，设计波音 747 这样一个大型的项目要花费 6 年的时间。但是，通过使用思维导图，他们的工程师只使用了 6 个月的时间就完成了波音 747 的设计，仅此一项，就帮助波音公司节省了 1 千万美元。公司将所有的飞机维修工作手册绘制成一张 25 英尺的思维导图，使原来需要花 1 年以上才能消化的数据，现在只要几周就可以了[①]。

概念图与思维导图是类似的概念绘制工具，但是仍然有差别。概念图突出概念结点和关系连线，而思维导图通常只受限于描绘阶层（树状）架构的关系。思维导图是使用一个关键词或想法引起形象化的构造和分类的想法；它用一个中央关键词或想法以辐射线形式连接所有的代表字词、想法、任务或其他关联项目（附图 A.5）。它可以利用不同的方式去表现人们的想法，如引题式，可见形象化式、建构系统式和分类式，普遍地用在研究、组织、解决问题和项目规划中。

（三）认知地图

认知地图（Cognitive Maps）是一种图形表征，它可以使认知主体对于特别的、可选择思想元素的理解得以可视化，锁定了人们与他们所面对的信息环境的关系。认知地图用图形方式将主体的逻辑和思想脉络表现出来，提供了一个关于主体全盘观点的图画，同时并不丧失细节，可以使研究者采取归纳式分析来提取和阐明所涌现出来的问题[②]。

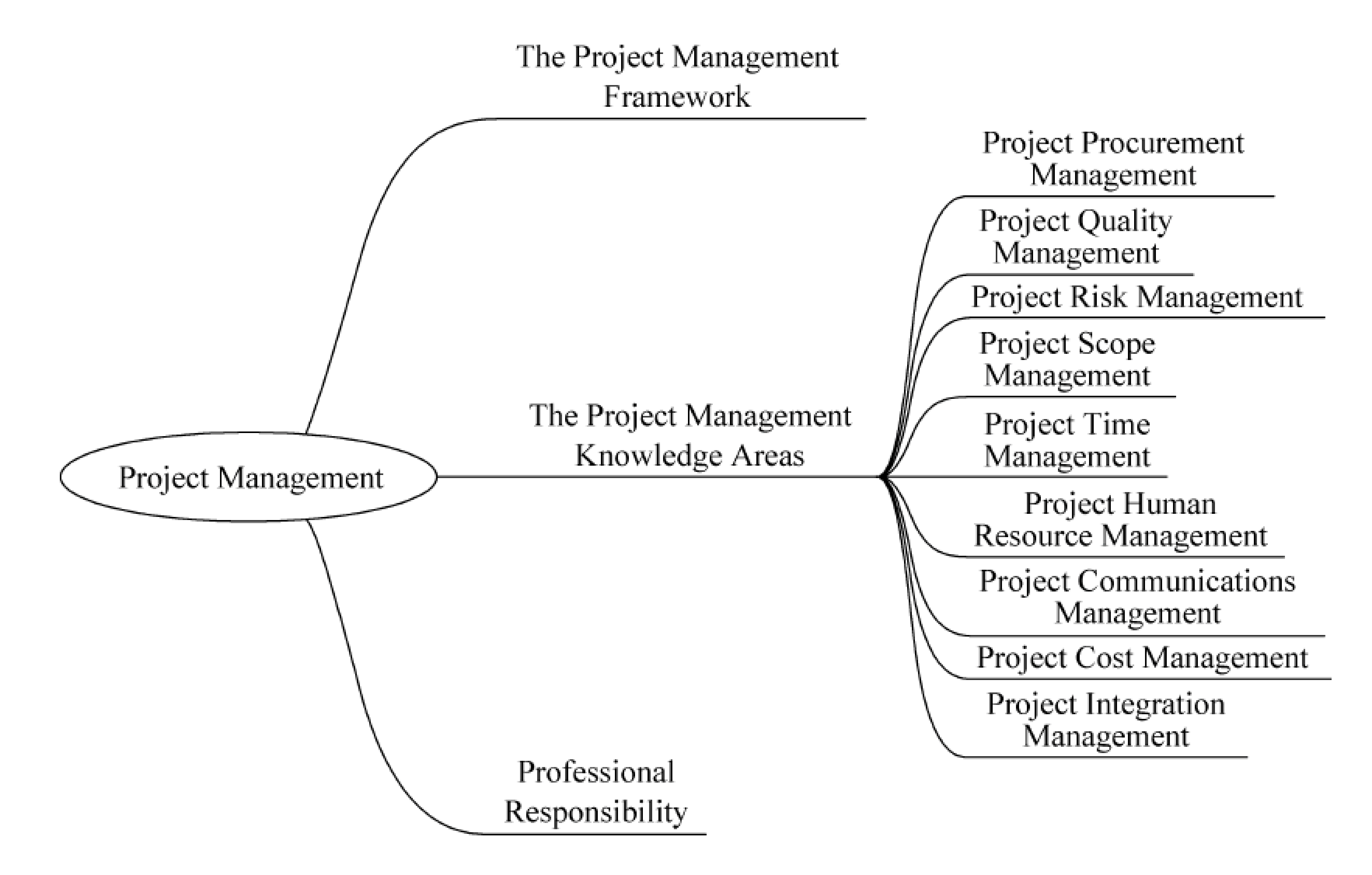
认知地图的一个最大特征就是很多结点之间都有着明显的因果性，这些因果性展示了主体的思维逻辑和行动过程，同时也使一系列分散的结点联系起来。而这些联系恰好为人们打开了了解主体思维秘密的大门。所以，认知地图也被称为因果地图（Causal Map）[③]。

---

① 王驰. 听“世界大脑先生”诠释“思维导图”. 中国妇女报，2005.02.03

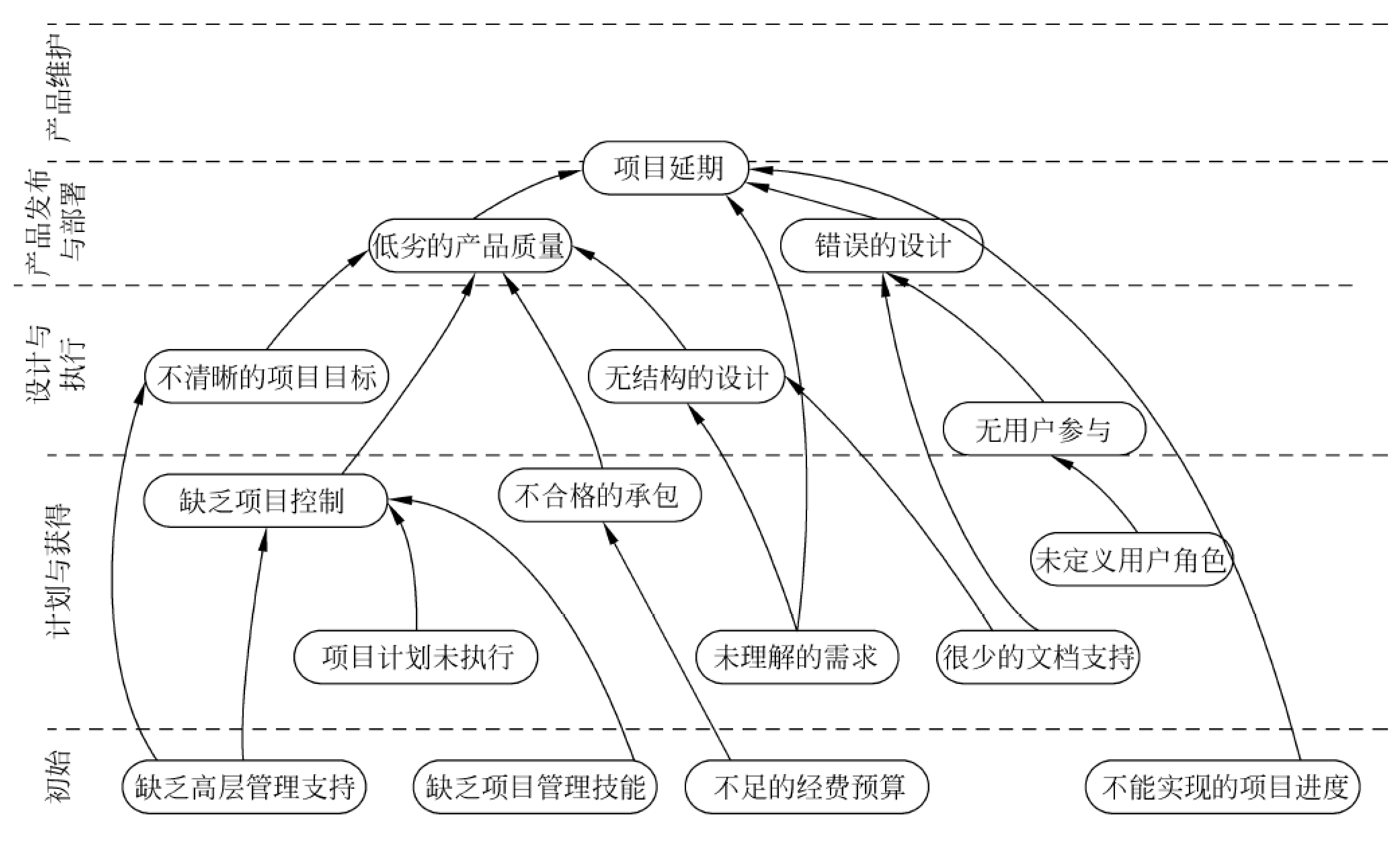
② 倪旭东，张钢. 作为思想挖掘工具的认知地图及其应用[J]. 科研管理，2008，29(4)：19～22

③ Eden C., Ackermann F.. Analyzing and comparing idiographic causal maps. Eden and Spender (Eds.), Managerial and organizational cognition [C], SAGE, London, 1998: 192～209



附图 A.5 用思维导图表示的项目管理

认知地图可以根据所掌握的概念、事件以及它们之间的关系来绘制，由于认知地图重在让人理解其中的逻辑关系，因此它的图形表达要力求简洁直观，一般就由结点和连线两部分组成。认知地图主要用于帮助人们规划工作，促进项目的决策，如附图 A.6 所示。



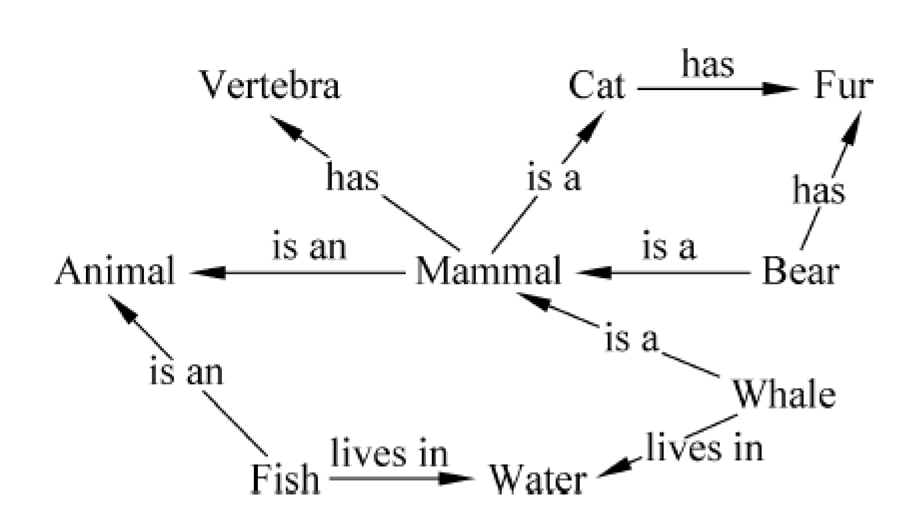
附图 A.6 造成项目延期的因果关系图

（四）语义网络

语义网络(Semantic Networks)于 1973 年由美国人工智能专家西蒙提出，是用于表示词与词之间的语义关系的一种网络理论。其原理是以句中词的概念为网络的结点，以沟通结点之间的有向弧来表示概念与概念之间的语义关系，构成一个彼此相连的网络，以理解自

然语言句子的语义。语义网络可用来对知识做出陈述性表示。它比逻辑表示直观，在用于常识推理时，有时也较为方便。

在语义网络中，带标号的结点表示思考对象——具体事物、抽象概念和状态等；带标号的有向弧则表示结点所代表对象间的关系。采用语义网络来理解自然语言时，首先分解输入句的句法关系，同时分析句子的深层结构，记录语义关系，最后求出输入句的语义网络，借以理解自然语言的语义。语义网络的示例如附图 A.7 所示。



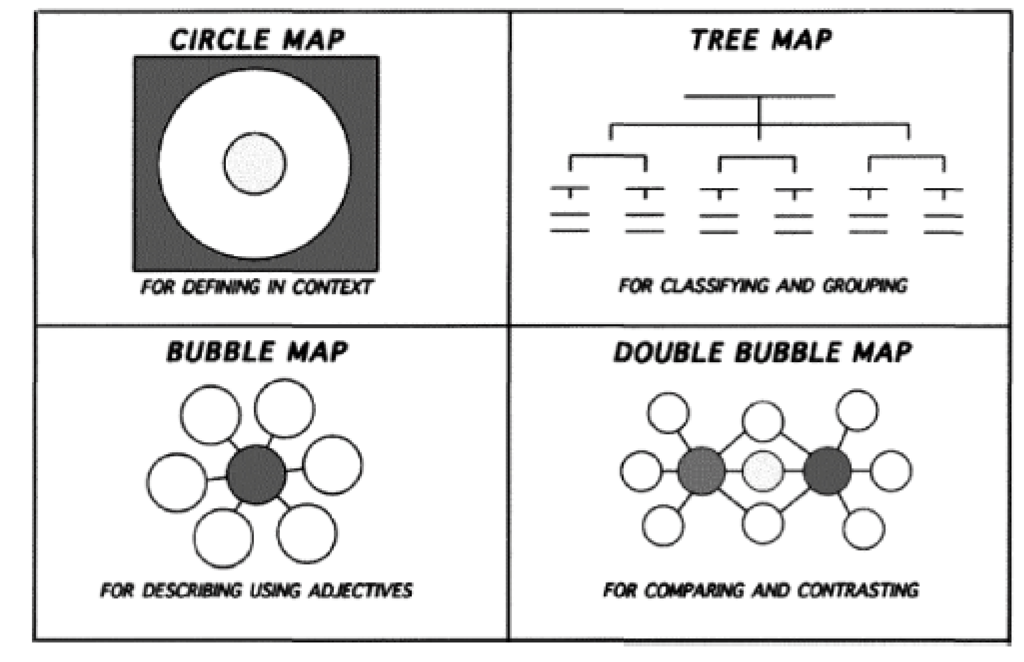
附图 A.7　哺乳动物的语义网络表示

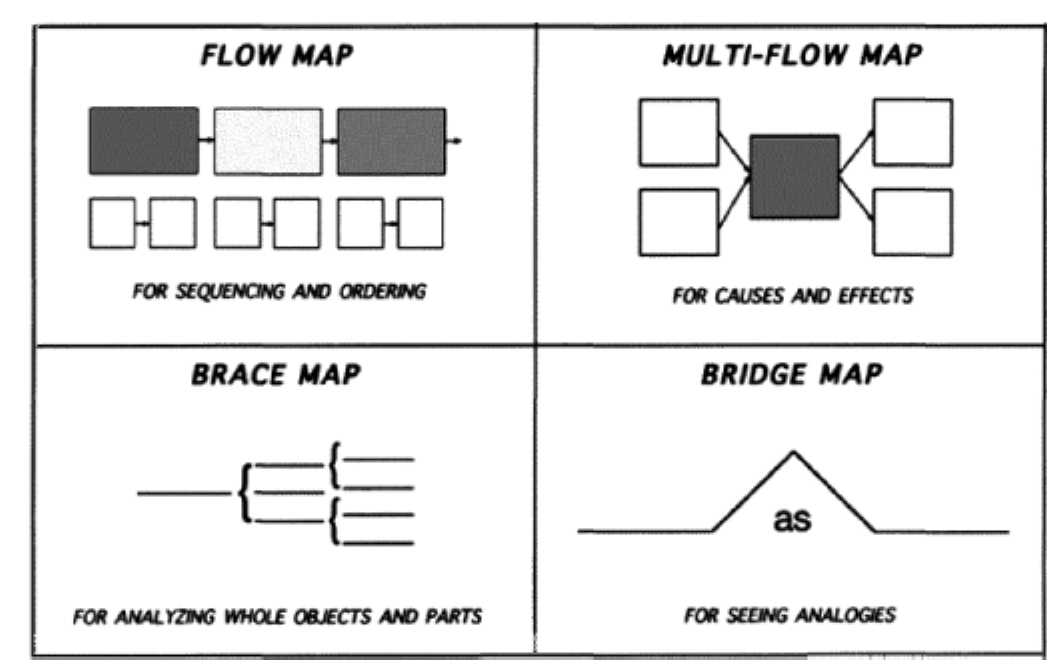
思维导图可以被认为是语义网络的一种自由变体。思维导图通过颜色以及图片生成一个语义网络（Semantic Net），从而激发人的创造力。

（五）思维地图

思维地图（Thinking Maps）是由 David Hyerle 博士 1988 年开发的帮助学习的语言，同样是信息时代下产生的一种知识可视化工具，在美国发起并广泛应用，后来被新西兰、新加坡等国家和地区引入，并取得了良好的效果。国内对“Thinking Maps”的翻译不同，多数译成“思维地图”，还有的译成“思想图”。

基于对大脑接收信息机制的研究，思维地图主要有 8 种类型，它们分别对应人在思考时的 8 种思维过程。思维地图提供了一套视觉模式进行思考，从而促进学生思考神经网络的发展，提高大脑不断认知和建构知识信息的能力，更突出培养学习者独立迁移思考的能力。这 8 种图分别是圆圈图（Circle Map）、起泡图（Bubble Map）、双起泡图（Double Bubble Map）、树形图（Tree Map）、括号图（Brace Map）、流程图（Flow Map）、复流程图（Multi-Flow Map）和桥形图（Bridge Map），这些图都是以基本的认知技巧为基础的，这些技巧包括比较和对比、排序、归类和因果推理等①。附图 A.8 所示为双起泡图的应用，它通常用于比较和对比的认知过程。





附图 A.8　思维地图的 8 种图式类型

① 赵国庆，陆志坚.“概念图”与“思维导图”辨析[J].中国电化教育，2004，(8)

（六）IDEF0 图表

IDEF 是 ICAM DEFinition method 的缩写，是美国空军在 20 世纪 70 年代末 80 年代初 ICAM（Integrated Computer Aided Manufacturing）工程在结构化分析和设计方法基础上发展的一套系统分析和设计方法。IDEF0 方法是其中的一个内容，在 ICAM 中用来建立加工制造业的体系结构模型，其基本内容是 SADT（System Analysis and Design Technology）的活动模型方法。

IDEF0 的基本思想是结构化分析方法，来源于 SADT 方法。它具有以下一组基本特色，这些特色形成一种思维规则，适用于从计划阶段到设计阶段的各种工作。

1. 全面地描述系统，通过建立模型来理解一个系统

一般地说，一个系统可以被认为是由对象物体（用数据表示）和活动（由人、计算机和软件来执行）以及它们之间的联系组成，至多只反映一个侧面，这样的技术很难说明系统的全貌。IDEF0 能同时表达系统的活动（用盒子表示）和数据流（用箭头表示）以及它们之间的联系。所以 IDEF0 模型能使人们全面描述系统。

IDEF0 图表示一种活动，是 IDEF0 最基本的元件，通常使用动词描述活动特性。箭头表示输入（Input），控制（Control），输出（Output），机制（Mechanisms），用于连接系统中各活动，通常是由名词描述。其结构如附图 A. 9 所示。

控制(C)
输入(I)
活动
Activity
输出(O)
机制(M)

附图 A. 9　IDEF0 图的基本结构

输入（Input）：实行或完成特定活动所需的资源，置于框图的左侧。

输出（Output）：经由活动处理或修正后的产出，置于框图的右侧。

控制（Control）：活动所需的条件限制，置于框图的上方。

机制（Mechanisms）：完成活动所需的工具，包括人员、设施及装备，置于框图的下方。

对于新的系统来说，IDEF0 能描述新系统的功能及需求，进而表达一个能符合需求及能完成功能的实现。对已有系统来说，IDEF0 能分析应用系统的工作目的、完成的功能及记录实现的机制。在这两种情况下都是通过建立一种 IDEF0 模型来体现的。模型就是系统的一种书面描述。它不一定必须用某种数学公式表示，可以是图形，甚至可以是文字叙述。因而可以说：“不管何种形式，只要 M 能回答有关实际对象 A 的所要研究的问题，就可以说 M 是 A 的模型”。

IDEF0 图形中同时考虑活动、信息及接口条件。它把方盒作为活动，用箭头表示数据及接口。因此在表示一种当前的操作，表示功能说明或设计时，总是由一个活动模型、一个信息模型及一个用户接口模型组成。

2. 目的与观点

IDEF0 要求在画出整个系统的功能模型时，具有明确的目的与观点。例如，对一个企业的 CIM 系统，必须有明确地站在厂长（或经理）的位置上建模的观点，所有不同层次的作者都要以全局的观点来进行建模工作，或者说就是为厂长而建模。这样才能保证是从全企业的高度来揭示各部分之间的相互联系和相互制约的关系。否则只从企业的各个业务部门的思维与利益考虑，则会陷入“只见树木不见森林”的风险。

3. 区别“什么”(What)和“如何”(How)

“什么”是指一个系统必须完成的是“什么”功能,“如何”是指系统为完成指定功能而应“如何”建立。就是说,在一个模型中应能明确地区别出功能与实现间的差别。

IDEF0 首先建立功能模型。把表示“这个问题是什么”的分析阶段,与“这个问题是如何处理与实现”的设计阶段仔细地区别开来。这样,在决定解法的细节之前,保证能完整而清晰地理解问题。这是系统成功开发的关键所在。

在设计阶段,要逐渐识别各种能用来实现所需功能的机制,识别选择适当机制的依据是设计经验及对性能约束的知识。根据不同模型,机制可以很抽象,也可以是很具体的。重要的是,机制指出了“什么”是“如何”地实现的。IDEF0 提供了一种记号,来表示在功能模型中如何提供一个机制来实现一个功能,及单个机制如何能在功能模型的几个不同地方完成有关功能。

有时机制相当复杂,以致机制本身需要进行功能分解。

4. 自顶向下分解

用严格的自顶向下地逐层分解的方式来构造模型,使其主要功能在顶层说明,然后分解得到逐层有明确范围的细节表示,每个模型在内部是完全一致的。

IDEF0 在建模一开始,先定义系统的内外关系,来龙去脉。用一个盒子及其接口箭头来表示,确定了系统范围。由于在顶层的单个方盒代表了整个系统,所以写在方盒中的说明性短语是比较一般的,抽象的。同样,接口箭头代表了整个系统对外界的全部接口。所以写在箭头旁边的标记也是一般的,抽象的。然后,把这个将系统当做单一模块的盒子分解成另一张图形。这张图形上有几个盒子,盒子间用箭头连接。这就是单个父模块所相对的各个子模块。这些分解得到的子模块,也是由盒子表示,其边界由接口箭头来确定。每一个子模块可以同样地细分得到更详细的细节。

相关课程教材推荐

ISBN	书　　名	定价(元)
9787302161837	嵌入式技术基础与实践	39.00
9787302161813	嵌入式技术基础与实践实验指导	19.00
9787302109693	J2ME 移动设备程序设计	29.00
9787302172574	计算机网络管理技术	25.00
9787302168003	计算机组成与系统结构	34.00
9787302109013	微机原理、汇编与接口技术	28.00
9787302142867	XML 实用技术教程	25.00
9787302167327	微机组成与组装技术及应用教程	29.50
9787302119715	计算机硬件技术基础	23.00
9787302147640	汇编语言程序设计教程(第 2 版)	28.00
9787302153542	DSPs 原理及应用教程	26.00
9787302146902	信号与系统(第二版)	34.00
9787302160670	电子技术	34.00

以上教材样书可以免费赠送给授课教师,如果需要,请发电子邮件与我们联系。

教学资源支持

敬爱的教师:

感谢您一直以来对清华版计算机教材的支持和爱护。为了配合本课程的教学需要,本教材配有配套的电子教案(素材),有需求的教师可以与我们联系,我们将向使用本教材进行教学的教师免费赠送电子教案(素材),希望有助于教学活动的开展。

相关信息请拨打电话 010-62776969 或发送电子邮件至 weijj@tup.tsinghua.edu.cn 咨询,也可以到清华大学出版社主页(http://www.tup.com.cn 或 http://www.tup.tsinghua.edu.cn)上查询和下载。

如果您在使用本教材的过程中遇到了什么问题,或者有相关教材出版计划,也请您发邮件或来信告诉我们,以便我们更好为您服务。

地址:北京市海淀区双清路学研大厦 A 座 708　计算机与信息分社魏江江　收
邮编:100084　电子邮件:weijj@tup.tsinghua.edu.cn
电话:010-62770175-4604　邮购电话:010-62786544